VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Deep learning v analýze obrazu

Deep Learning of Image Analysis

2018 David Kuchař

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Kuchař**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Deep learning v analýze obrazu
Deep Learning of Image Analysis

Jazyk vypracování: čeština

Zásady pro vypracování:

Deep learningu je v poslední době věnována pozornost ve výzkumu i v praxi, které se využívají na klasifikaci. Dosahuje zajímavých detekčních výsledků, přičemž zpracování lze provádět i na GPU a tím ušetřit čas. Student řádně nastuduje funkci konvolučních neuronových sítí a sestaví ve frameworku síť GoogleNet[4], kterou odzkouší na zvoleném data setu (například http://www.image-net.org/) pro klasifikaci objektů.

1. Seznamte se s metodami používanými v Deep learningu jako jsou konvoluční neuronové sítě a náležitě je popište.
2. Seznamte se s existujícími frameworky (Caffe, Theano, TensorFlow, DLib,...) pro praktickou implementaci a popište jejich dostupnost, využití apod.
3. Sestavte a odzkoušejte konvoluční neuronovou síť s využitím frameworků Caffe a TensorFlow (případně si zvolte jiný framework).
4. Zjištěné poznatky řádně zdokumentujte v textu práce.

Seznam doporučené odborné literatury:

[1] Chapter 10. Neural Networks. The Nature of Code [online]. Dostupné z: http://natureofcode.com/book/chapter-10-neural-networks/
[2] JUERGEN, Schmidhuber. Deep Learning in Neural Networks: An Overview. ArXiv [online]. 2015, 88 DOI: 10.1016/j.neunet.2014.09.003. Dostupné z: http://arxiv.org/abs/1404.7828
[3] Convolutional Neural Networks (LeNet). Deep Learning: . . . moving beyond shallow machine learning since 2006! [online]. 2016 Dostupné z: http://deeplearning.net/tutorial/lenet.html
[4] SZEGEDY, Christian, WEI LIU, YANGQING JIA, et al. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [online]. IEEE, 2015, s. 1-9 [cit. 2017-10-12]. DOI: 10.1109/CVPR.2015.7298594. ISBN 978-1-4673-6964-0. Dostupné z: http://ieeexplore.ieee.org/document/7298594/
[5] LECUN, Yann, Léon BOTTOU, Yoshua BENGIO a Patrick HAFFNER. Gradient-Based Learning Applied to Document Recognition [online]. 1998, 46 Dostupné z: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radek Simkanič, DiS**

Datum zadání: 01.09.2017

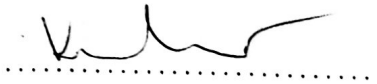
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2018



Rád bych poděkoval vedoucímu bakalářské práce Ing. Radkovi Simkaničovi, DiS. za jeho cenné rady a připomínky při tvorbě. Dále také děkuji své rodině a přátelům, kteří mě po celou dobu studia podporují.

Abstrakt

Tématem této bakalářské práce je deep learning v analýze obrazu. V práci jsou popisovány principy neuronových a konvolučních neuronových sítí s dnešními nejpoužívanějšími frameworky. Následuje praktické seznámení s tvorbou konvolučních sítí v frameworku TensorFlow a Pytorch a jejich demonstrace. Využit byl k tomu objektově-orientovaný skriptovací jazyk Python s frameworky TensorFlow a Pytorch využívající platformu CUDA.

Klíčová slova: Strojové učení, neuronové sítě, CNN, Frameworky, GPU, TensorFlow, Pytorch, Python, CUDA, GoogLeNet

Abstract

Theme of this bacherlogy thesis is deep learning in image analysis. The thesis description principles of neural and convolution Neural Networksk with today most used framework, followed by practical skill with build of convultation network in TensorFlow and Pytorch framework and their demonstrate. This work is based on object-soritented and scripting programming language Python with framework TensorFlow and Pytorch work on CUDA platform.

Key Words: Deep learning, neural network, CNN, Frameworks, GPU, TensorFlow, Pytorch, Python, CUDA, GoogLeNet

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface - je v informatice rozhraní pro programování aplikací
BAIR	–	Berkeley AI Research - společnost vyvíjející Caffe
CIFAR10	–	Canadian Institute For Advanced Research - Kanadský institut pro pokročilý výzkum
CNN	–	Convolution Neural Network - konvoluční neuronová síť
CNTK	–	Microsoft Cognitive Toolkit - framework pro tvorbu neuronových sítí
CPU	–	Graphic Processing Unit - grafický procesor neboli grafická karta
cuBLAS	–	CUDA Basic Linear Algebra Subroutine library - Základní lineární algebraická knihovna CUDA
CUDA	–	Compute Unified Device Architecture - hardwarová a softwarová architektura od společnosti nVIDIA, která umožňuje na GPU spouštět programy
cuDNN	–	Cuda Deep Neural Network libray - knihovna na hlubkové neuronové sítě založené na CUDA
ČI	–	Čas iterace
GPU	–	Central Processing Unit - centrální procesorová jednotka nebo-li procesor
HL	–	Hodnota loss
ILSVRC	–	The ImageNet Large Scale Visual Recognition Challenge - soutěž vrozpoznávání z obrazů
KVU	–	Konečný výsledek učení
MCP	–	McCulloch-Pitts - druh preceptronu
MLP	–	Multi Layered Perceptron - vícevrstvá neuronová síť perceptronů
MNIST	–	Mixed National Institute of Standards and Technology database - databáze národního institutu pro standardy a technologie v USA
NCCL	–	NVIDIA Collective Communications Library - Knihovna kolektivní komunikace NVIDIA
PČ	–	Průměrný čas
PHL	–	Průměrná hodnota loss
PV	–	Průměrný výsledek
RBF	–	Radial Basis Functions - typ třívrstvé neuronové sítě
ReLU	–	Rectified linear unit - aktivační přenosový funkce
RNN	–	Recurrent Neural Network - rekurentní neuronová síť
SDK	–	Software development kit - systémový vývojový nástroj

TF	–	TensorFlow - framework na tvorbu CNN sítě

Seznam obrázků

Seznam tabulek

1 Úvod

Deep learning je obor strojového učení (machine learning) a umělé inteligence založené na nejrůznějších algoritmech. Deep learning se snaží co nejvíce přiblížit lidskému učení.

Klasifikace objektu z obrazu je v dnešní době velmi rozšířená, ať už se jedná o detekci lidské tváře, klasifikace ručně psaných znaků nebo jiných objektů z obrázků. Pro tuto realizaci se velice často používají implementace pomocí různých typů neuronových sítí, které jsou díky své architektuře schopny se učit rozpoznávání různých vzorů a na základě toho vyhodnocovat objekty na obrazu.

V dnešní době jsou neuronové sítě velice oblíbené. Jejich masivní narůst zapříčinila zejména paralelizace jak na CPU, tak GPU. S neuronovými sítěmi se setkáváme ve spoustě odvětví - v analýze obrazu, medicíně, hrách, zpracování ekonomických dat apod.

Mezi jedny z neuronových sítí v deep learningu se řadí i tzv. konvoluční neuronová síť, která se nejčastěji používá k analýze obrazu a segmentaci obrazu. Díky jednoduchosti a téměř žádné nutnosti předchozí úpravy obrázku se stala oblíbenou neuronovou sítí.

Dnes již pro deep learning existují nejrůznější frameworky, které nám usnadňují tvorbu neuronových sítí, sledují aktuální trendy a reagují na ně. Ať už ve zpracování sítě za pomocí GPU nebo tvoří dynamickou neuronovou síť.

Cílem práce je popis současných možností neuronových sítí, konvolučních sítí a frameworků umožnující nám jejich tvorbu. Testování obsahuje i algoritmy či knihovny běžící na platformě CUDA, které jsou určeny ke zpracování velkému objemu dat. Teoretická část je zaměřená na seznámení neuronových a konvolučních sítí a dnešní nejpoužívanější frameworky. V teoretické části jsou pak demonstrované konvoluční sítě tvořené ve frameworku TensorFlow a Pytorch a jejich tvorba. V závěru se práce zabývá rozdíly a vyhodnocením jednotlivých frameworků.

2 Neuronové sítě

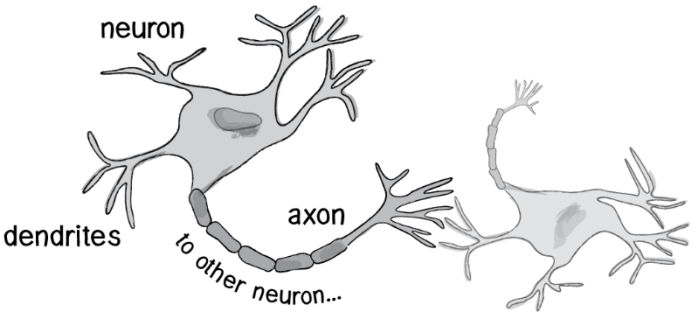
Teorie neuronových sítí je založena na poznatcích o zpracování informací v nervových buňkách. Světoví odborníci byli dlouhou dobu inspirovaní funkcí lidského mozku, od doby kdy v roce 1943 neurofyziolog Warren S. McCulloch a Walter Pillts vyvinuli první konceptuální model umělé neuronové sítě. Jejich práce nebyla přímo určená k tomu, jak přesně biologický mozek funguje, ale jak pomocí umělé neuronové sítě řešit určité druhy problémů. Existují problémy, s kterými si počítač poradí velice snadno např. výpočty nebo vzorce. Naopak jsou zde i problémy s kterými si člověk poradí, ale počítač nikoliv. Například rozpoznávání vzorů nebo lidských tváři. Pro tyto případy se začaly využívat umělé neuronové sítě. [1]

2.1 Biologický neuron

Neurony jsou buňky v živých organismech, které umožňují vést signály a reagovat na ně. Každý neuron má několik dendritů, které mají za úkol přijímat signál z okolí. Jeden výběžek tzv. axon je schopný signál vyslat. Axon každého neuronu je zakončen tzv. synapsemi, které dosedají na jiné neurony. Schéma biologického neuronu na obrázku 1. Přes synapse se přenášejí vzruchy mezi neurony. Rychlost přenosu je v rozmezí $5 - 125$ m/s. [2]

Nervová soustava člověka obsahuje cca $10^{11} - 10^{12}$ neuronů. S rostoucím věkem počet neuronů ubývá. Neurony mohou mít desítky, ale někdy i stovky až tisíce dendritů, které jsou vzájemně propojeny. Pokud odumře velká část neuronů, ztrátu lze nahradit zvětšením počtu spojů – dendritů. [1, 2]

Pomocí dendritů neurony zachycují signály. Tyto signály se šíří dovnitř buňky, kde vzniká potenciál. Je-li tento potenciál dostatečně velký, neuron je schopen sám vyslat signál dál. [2]



Obrázek 1: Schéma biologického neuronu. [1]

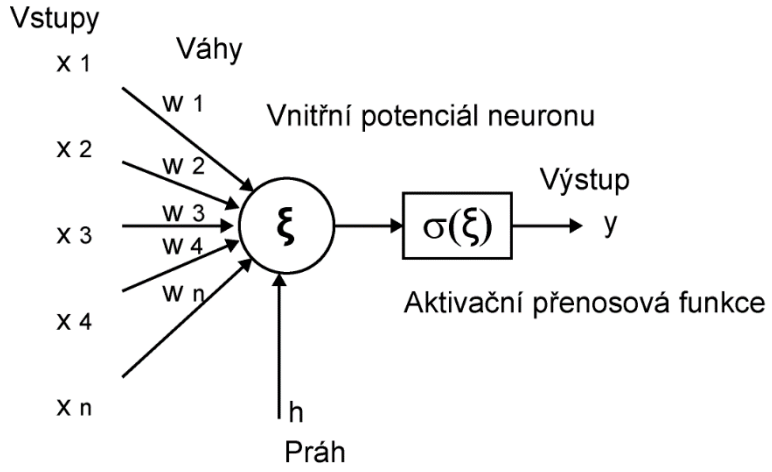
2.2 Umělý neuron

Model umělého neuronu (obrázek 2) představuje z větší části abstraktní mechanizmus, jak nervové buňky zpracovávají informace. Nelze totiž vytvořit přesnou analogii modelu skutečného neuronu. Proto modely umělých neuronů, které se v současnosti používají, kopírují pouze základní funkci neuronu. [1]

Klasický umělý neuron McCulloch-Pitts (MCP), nebo také perceptron, je tvořen několika vstupy $x_1 - x_n$ a jedním výstupem y. Každý vstup má svou vlastní váhu, která muže být i záporná. Vážená suma vstupních hodnot představuje vnitřní potenciál neuronu, který lze vypočítat podle vztahu:

$$y = bias + \sum_{i=1}^{n} x_i \cdot w_i \tag{1}$$

kde x_1 až x_n jsou vstupy neuronu, w_1 až w_n je vektor vah jednotlivých vstupů a $bias$ (práh) je konstantní vstup neuronu. Z praktických důvodů se práh modeluje jako jedna z vah tak, že vstupní vektor i vektor vah je rozšířen o nultou pozici. Vstup na nulté pozici je vždy považován za roven 1 a nultá váha je nastavena na hodnotu h (práh). V takovém případě se práh stává jednou z vah a v průběhu trénování podléhá adaptaci. [2, 3]

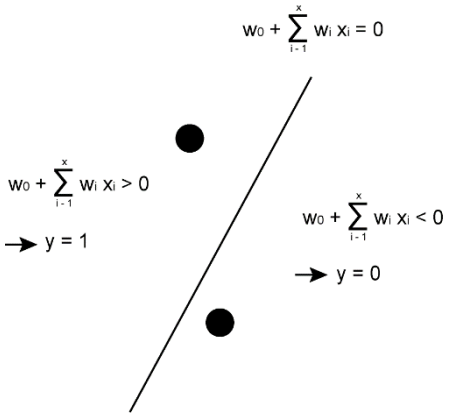


Obrázek 2: Schéma umělého neuronu. [3]

Pokud potenciál neuronu má dostatečně velkou hodnotu, vyšle signál. Aktivační přenosová funkce se obecně používá jako nelineární funkce transformující hodnotu vnitřního potenciálu neuronu - nejčastěji sigmoid. Pro ilustraci však předpokládejme, že použijeme nejjednodušší nelineární typ, ostrou nelinearitu, kdy platí:

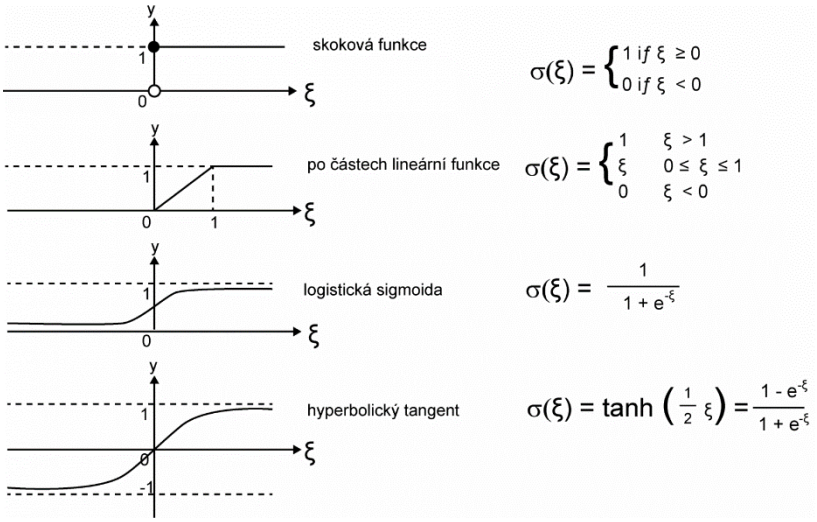
$$\sigma(\varsigma) = \begin{cases} 1 : \varsigma \geq 0 \\ 0 \end{cases}$$

Předpokládejme neuron pro $n = 2$ se dvěma reálnými vstupy x_1 a x_2 a váhami w_1 a w_2. Takto definovaný neuron $\{R\}^2 - > 0, 1$. Neuron ve své aktivní dynamice reaguje na vstupy a přiřazuje jim hodnotu 0 nebo 1. Provádí tak klasifikaci těchto bodů do dvou skupin podle hodnoty aktivační funkce tzv. výstup neuronu. V tomto konkrétním případě je zařazení bodu dáno jejich pozicí vůči přímce definované aktivační funkcí (váhou neuronu). Dělící přímka rozděluje dvojrozměrný prostor na dvě skupiny (viz obrázek 3). [1, 3]



Obrázek 3: Klasifikace neuronu v rovině. [3]

Aktivační přenosové funkce nejčastěji používané v neuronových sítí lze vidět na obrázku 4. Existují i odlišné koncepty neuronu, kde výstup neuron je kalkulován jiným způsobem např. sítě na radiální bázi (RBF sítě), zde neuron vyčísluje vzdálenost vstupního vektoru x_{id} vektoru vah w nebo také vlnové sítě (Elliott wave neural network). [2, 3]



Obrázek 4: Aktivační přenosové funkce. [3]

Hlavní vlastností neuronových sítě je schopnost se učit. Neuronová síť je složitý systém, který je schopný se vyvíjet dle informací, které zpracovává. Jsou-li výsledky sítě dobré, váhy neuronu se nezmění, v jiném případě se váhy pozmění tak, aby výsledky vycházely lépe. [1]

Máme několik kategorií učení:

- **Učení s učitelem (Supervised Learning)** – při učení neuronové sítě jsou předkládány požadované výsledky a srovnávány s výstupem neuronové sítě. Podle rozdílů se pak určuje další kola učení.

- **Učení bez učitele (Unsupervised Learning)** – není založeno na vyhodnocování výstupu. Síť se sama snaží třídit vstupy podle charakteristických znaků a podobnosti
- **Kombinace učení s učitelem a bez něj (Semi Supervised Learning)** - část výstupu je porovnána s požadovaným výstupem. Další data jsou poté vyhodnocena bez známého výstupu.
- **Zpětnovazebné učení (Reinforcement Learning)** – je založené na předchozích zkušenostech a okolnostech. Například když robot narazí, zapamatuje si, že tudy neprojede. [3]

2.3 Umělá neuronová síť

Každá umělá neuronová síť se skládá ze vzájemně propojených neuronů, takže výstup u každého neuronu je také vstup do jiných neuronů. Počet neuronů a jejich vzájemné propojení určuje topologie sítě. [1]

Topologii rozdělujeme na dva základní typy:

- Neuronovou síť s dopředným šířením signálu (feedforward)
- Neuronová síť se zpětnovazebním šířením signálu (feedback)

U neuronové sítě s dopředným šířením postupují všechny signály ve směru ze vstupní vrstvy do vrstvy výstupní bez zpětné vazby. Zatímco u sítí se zpětnovazebním šířením signálu je vstup každého neuronu závislý na hodnotě výstupu z předchozího cyklu.

Neurony jsou strukturované do vrstev. Každá neuronová síť se skládá minimálně ze dvou vrstev a to vstupní a výstupní, případně z dalších vrstev, které jsou tzv. skryté. Cílem vstupní vrstvy je zabezpečit distribuci vstupních signálů sítě do ostatních vrstev. Neuron v ní má pouze jeden vstup a posílá vstupní signál na výstupní beze změny. Výstupní vrstva určuje výstup neuronové sítě. [1, 2, 4]

3 Modely neuronové sítě

3.1 Perceptron

Perceptron je nejednodušší model neuronové sítě, kterou v roce 1957 vyvinul Frank Rossenblatt. Prvotně byl navržený jako model zrakové soustavy. V dnešní době se často nazývá neuron. Nejzákladnější úlohou, kterou řeší Perceptron je klasifikace. Například klasifikace výrobku do dvou skupin. Později však došli k závěru, že Perceptron nemá tak široké využití kvůli použitelnosti pouze v klasifikaci lineárně separovaných skupin. Proto došlo k rozšíření na vícevrstvý perceptron MLP (Multi Layered Perceptron). [1]

Perceptron se typicky skládá z n vstupů, procesoru a jednoho výstupu. Každý vstup je vynásoben váhou, která je mezi hodnotami $< -1, 1 >$. Tyto vážené vstupy se následně sečtou a vyhodnotí v aplikační funkci, která poté vyšle výstupní signál. Například vyhodnocení zda je číslo kladné či záporné. Je-li kladné, vyšle 1 jinak -1. [1, 3]

Při učení perceptronu se používá metoda učení s učitelem - Hebbova metoda pro adaptaci perceptronu, ve které je navrženo učící pravidlo. Do perceptronu se posílá údaj o očekávaném výsledku. Jsou-li výsledky odlišné, aktivuje se funkce, která pozmění váhy u vstupů podle velikosti chyby. Ta se definuje jako rozdíl mezi očekávaným a vypočteným výsledkem. Pokud jsou výsledky shodné, výsledek chyby je 0, je-li výpočet -1 a očekávaný výsledek $+1$, chyba je -2. Chyby jsou vyobrazeny v tabulce 1. [1]

Tabulka 1: Tabulka chyb

Očekávaný	Pravý	Chyba
-1	-1	0
-1	$+1$	-2
$+2$	-1	$+2$
$+2$	$+1$	0

Vzorec pro výpočet chyby:

$$chyba = očekávaný - pravý \tag{2}$$

Základní vzorec pro novou váhu pro vstup:

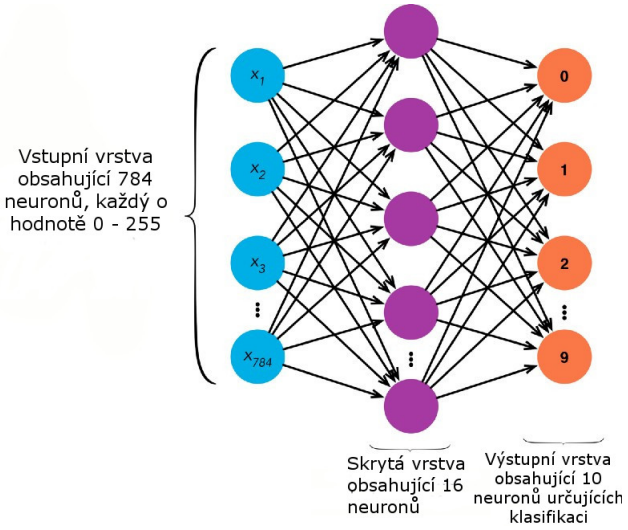
$$vaha_{t+1} = vaha_t + chyba_t \cdot vstupt \tag{3}$$

Základní vzorec pro novou váhu pro vstup se zohledněním rychlosti učení:

$$vaha_{t+1} = vaha_t + chyba_t \cdot vstupt \cdot ucebni\_konstanta \tag{4}$$

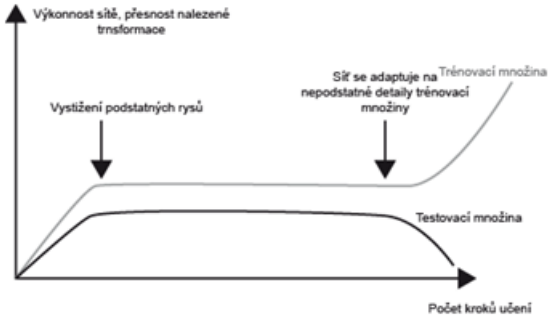
3.2 Vícevrstvá neuronová síť

Vícevrstvá neuronová síť je tvořená jednou nebo více skrytými vrstvami, těmto sítím se říká hluboké neuronové sítě. Má potenciál řešit nelineární problémy. K určení počtu vrstev nerunové sítě neexistuje žádné pevné pravidlo, je nutné tento problém řešit experimentálně. Existuje však několik pravidel, které se nám snaží tvorbu sítě ulehčit. [23, 25]



Obrázek 5: Vícevrstvá neuronová síť pro zpracovaní datasetu MNIST. Obrázek v plné velikosti v příloze. [23]

Téměř vždy stačí pro řešení problému jedna vnitřní vrstva. Topologie se dvěma vnitřními vrstvami může být zapotřebí, pokud se má neuronová síť naučit nespojitou funkci. Vetší počet skrytých vrstev poté nemá žádná doporučení, avšak muže se stát, že síť s více vrstvami o menším počtu neuronu bude efektivnější, než síť s menším počtem skrytých vrstev s více neurony na jedné vrstvě. Na obrázku 5, lze vidět návrh neuronové sítě pro vyhodnocení obrázku z MNIST, kdy je vstupní vrstva složená z 784 neuronu, což znamená, že existuje jeden neuron pro každý pixel obrázku o rozměrech 28×28 pixelů. [23]

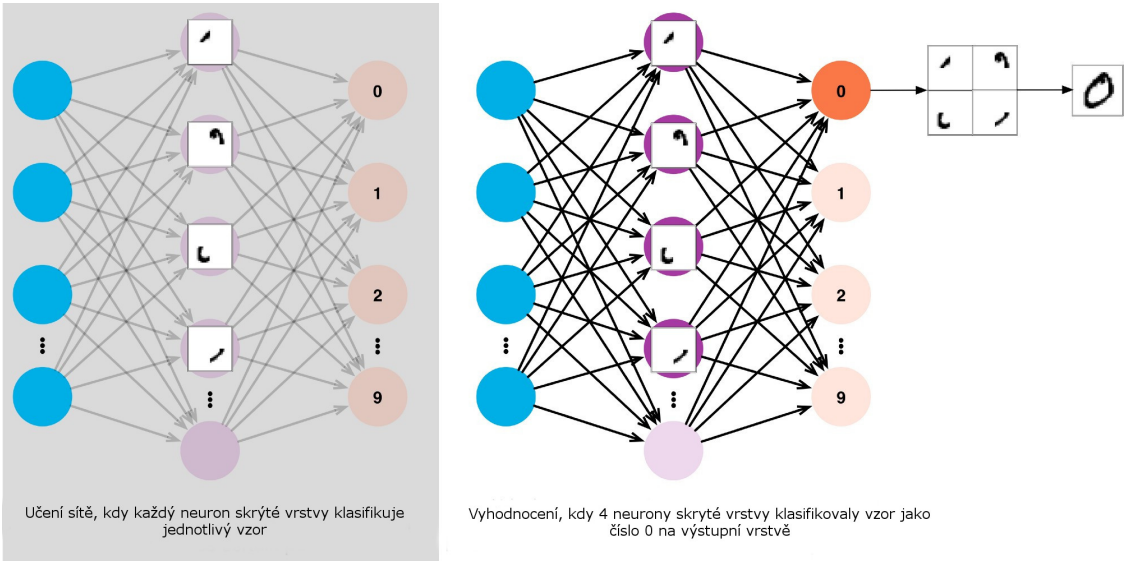


Obrázek 6: Syndrom přeučení. [27]

Správný počet neuronů ve vrstvách je velice důležitý. Při malém počtu neuronů nemá síť dostatečnou kapacitu k vyřešení problému. Naopak s velkým počtem neuronů dochází k delší době učení a v nejhorším případě i přeučení neuronové sítě. [23]

K přeučení dochází, když síť disponuje velkou kapacitu na zpracování informací. Je to stav, kdy se síť příliš přesně naučí množinu testovacích dat a to včetně náhodných chyb nebo šumu. Výsledky s trénovacími daty bývají vysoké, zatím co při práci s testovacími daty jsou výsledky tristní. Tento problém je demonstrován na obrázku 6. [23]

Při určovaní počtu neuronu se postupuje tak, že se začne menším počtem. Funkci určující výpočet chybovosti, pak je určeno, jak přesně je síť naučená a dle toho lze zvyšovat nebo snižovat neurony do doby, kdy chybovost klesne pod přijatelnou mez. Jak vyhodnocuje vícevrstvá neuronová sítě je znázorněno na obrázku 7.



Obrázek 7: Ukázka zpracování a vyhodnocení vícevrstvé neuronové sítě. Obrázek v plné velikosti v příloze. [27]

3.3 Feed forward sítě

Neuron je schopen rozdělit prostor pouze na dva poloprostory (viz kapitola 3.1). Ve složitějších úlohách je však rozdělení do dvou poloprostorů nedostačující. Proto se sestavují rozsáhlé neuronové sítě, složené z několika vrstev neuronové sítě vzájemně n propojených neuronů, kdy každý neuron je propojen s každým neuronem následující vrstvy, viz. obrázek 5. [6]

Každý neuron řeší pouze část problému. Výsledek je poté kompozicí všech výstupních funkcí neuronu v síti. Přístup v přírodě je velmi podobný, pokud by nějaký neuron nefungoval z jakýchkoliv důvodů, výsledek bude i tak správný, protože celková informace je rozprostřena mezi více neuronů.

Každý neuron rozděluje prostor na dva podprostory. První vrstva tedy rozdělí prostor na několik polorovin. Výstup lze popsat více detailně. Druhá vrstva už nepřijímá příznaky, ale příslušnost k těmto rovinám. Třídy tedy mohou být rozděleny nejen přímkami, ale i křivkami a celý výsledek je proto přesnější.

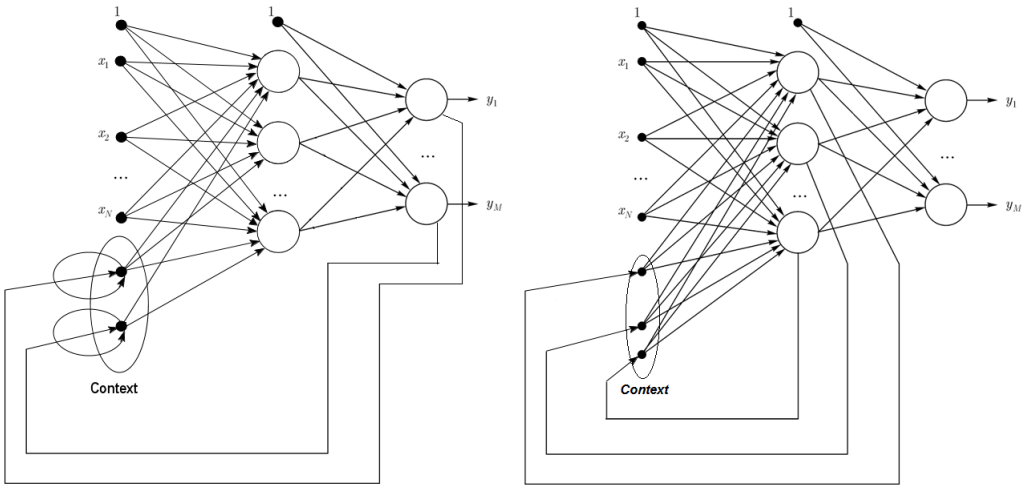
Nejčastěji se pro učení používá metoda BackPropagation, která funguje na porovnání s očekávaným výsledkem a podle chyby upraví váhy neuronu.

3.4 Rekurentní sítě

Rekurentní sítě - RNN (Recurrent Neural Network) existuje spoustu typů těchto sítí, jejíž hlavním rozdílem oproti dopředných sítí je, že výstup není dán pouze aktuálním vstupem, ale i vnitřním stavem sítě. Protože kóduje historii vstupů, které již byly zpracované, umožňuje jí to učit se různé reprezentace a závislosti mezi vstupy napříč časem. [11]

Za stav neuronu v určitém časovém okamžiku je považován jeho výstup. Ten je pak se zpožděním veden zpět na vstup dalších (klidně i stejných) neuronů. Tím je zajištěna návaznost na předchozí stavy. [24]

Plně rekurentní síť je základní architekturou RNN vyvinutá v roce 1980. V této topologii jsou předchozí vyhodnocení neuronů posíláno na vstup všem neuronům v síti.

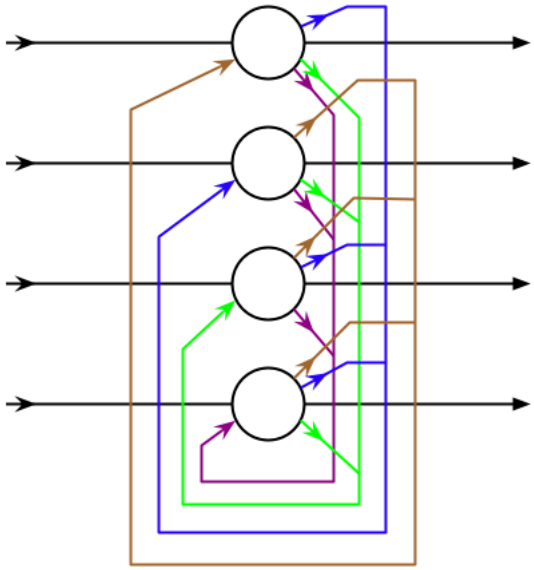


Obrázek 8: Vlevo Jordanova síť a vpravo Elmanova síť. [26]

Nejzákladnější RNN sítí je Jordanová a Elmanová sít (obrázek 8), jsou tvořeny třívrstvou feed forward sítí. Zpětná vazba v Elmanové síti pak vytváří kontext z výstupu skryté vrstvy. Počet kontextových neuronů je tak stejný jako počet neuronu ve skryté vrstvě. Zatímco Jordanová síť má kontext až z výstupní vrstvy neuronové sítě a tedy počet kontextových neuronu je totožný s výstupním počtem. [24, 25]

3.5 Hopfieldova síť

Hopfieldová neuronová síť byla vytvořena Johnem Hopfieldem v roce 1982. Vycházel z energetické funkce, z které odvodil pravidla pro učení a vybavování. Síť je tvořena n neuronů, z nichž každý je zároveň vstupem i výstupem sítě. To znamená, že všechny prvky jsou vzájemně propojeny mezi sebou.Schéma Hopfieldovy sítě obrázek 9. [4, 5, 8]



Obrázek 9: Hopfieldova síť. [9]

Výstup sítě je tvořen pouze $+1$ nebo -1, kde $+1$ je nastavena v případě, že součet vah je větší nebo rovno 0. V opačném případě je výstup nastaven na -1. [4]

$$u(x) = \begin{cases} 1 : \Sigma_i w_i x_i \geq 0 \\ 0 : \Sigma_i w_i x_i < 0 \end{cases}$$

Aktualizace vah je tvořena dvěma způsoby:

- **Asynchronní** - Aktualizován je pouze jeden neuron v jeden čas. Výběr jednotlivých neuronů je buď náhodný nebo v určitém pořadí.
- **Synchronní** - Všechny neurony jsou aktualizovány ve stejném čase. [9]

Hopfieldova síť má pro každý stav sítě přiřazenou skalární hodnotu E jako "energie". Cílem je dosáhnout stavu s nejmenší energií. Energie se vypočítá následujícím vztahem:

$$E = -\frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{n} W_{ij}x_i x_j \tag{5}$$

kde n udává počet neuronů sítě, W váhu vstupů a x hodnotu i-tého vstupu. Prahová hodnota se volí obvykle nulová. Energetická funkce popisuje, jaké chyby se síť dopouští použitím současných výsledků sítě.

Jednotlivé aktualizace sítě zajišťují, že je energie jednotlivých stavů stejná nebo menší jako při předchozím stavu sítě. [4, 8]

Používá se nejčastěji jako asociativní paměť (vybavování informace na základě její částečné znalosti) nebo klasifikátor pro rozpoznání obrazů. U Hopfieldové sítě jsou některá omezení, kdy síť nefunguje dle očekávání. Jedním z nich je omezený počet vzorů, které si síť pamatuje. Dalším problémem pak můžou být dva stavy se stejnou hodnotou energie. [10]

4 Konvoluční neuronové sítě

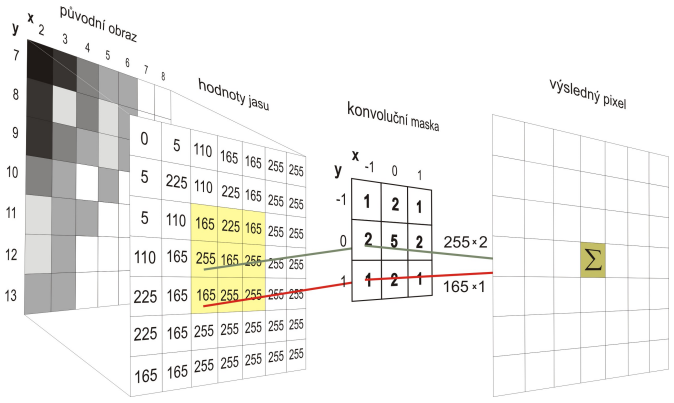
Konvoluční neuronové sítě - CNN (Convolutional Neural Network) jsou speciální vícevrstvé neuronové sítě, které se používají pro rozpoznávání vzoru v pixelovém obrázku s použitím minimálního preprocessingu. Nejčastěji se využívají pro klasifikaci obrazu (popřípadě vícerozměrných dat). [11, 12]

4.1 Charakteristika

Cílem této sítě je identifikace objektu na základě jejich podobnostních vzorů bez ohledu na deformaci, změnu velikosti nebo posunutí.

Typická konvoluční síť je na obrázku 11. Skládá se z několika vrstev. K charakteristickým vlastnostem konvoluční neuronové sítě patří získávání příznaků, sdílení vah a vzorkování.

První vrstvou je *konvoluční vrstva*. Na vstupu vrstvy je přiveden obrázek, na který je aplikován filtr, jehož úkolem je získání příznaků jako jsou hrany, rohy atd. Tento filtr je dále aplikován na celý obrázek po částech, vyobrazeno na obrázku 10. Vetšinou bývá v konvoluční vrstvě na jeden obrázek použito více druhu těchto filtrů. Výstup takto aplikovaného filtru je pak příznaková mapa. Tyto mapy se poté nacházejí na výstupu vrstvy v množství počtu použitých filtrů. Na ukázkovém obrázku 11 bylo aplikováno v první konvoluční vrstvě 16 filtrů a výstupem bude tedy 16 příznakových map (channels). [4]



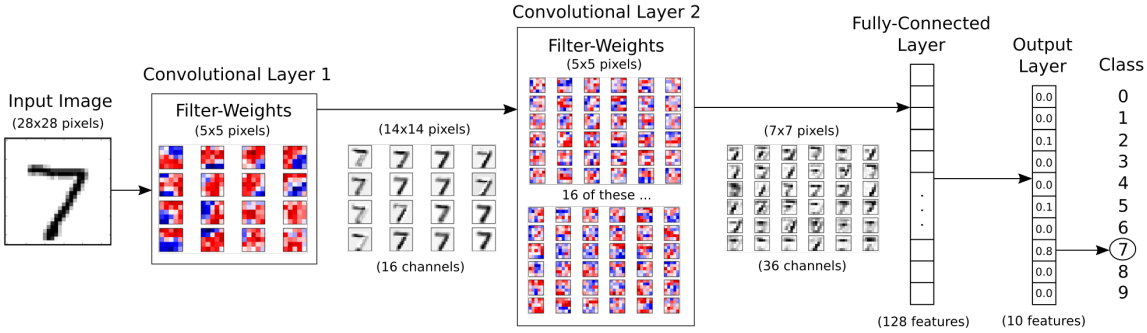
Obrázek 10: Aplikace filtru na obrázek. [12]

Z důvodu zrychlení a invariance (neměnný stav) se využívá vrstva škálovací (pooling layer). Cílem vrstvy je zredukovat velikost vstupu. Použije se na všechny příznakové mapy, které jsou výstupem konvoluční vrstvy. Existuje spousta funkcí, které škalovací vrstva využívá. V současnosti je to *max-pooling* (obrázek 17), která vezme na vstup oblast $k \times k$ a následně vybere jako výstup nejvyšší hodnotu v zadané oblasti.

Nejčastěji velikosti oblasti 2×2, která se vždy posouvá o 2 pixely a z každého regionu vybere nejvýraznější pixel. V důsledku aplikace filtru na každý druhý pixel se výška a šířka zmenší na polovinu a dochází tak k redukci dat. Protože v této vrstvě dochází ke ztrátě dat na úkor

výkonu, v některých moderních aplikacích je tato vrstva nahrazována odpovídající konvoluční vrstvou, která se naučí vhodný škálovací filtr.

Po aplikaci několika konvolučních a následné škálovacích vrstev je třeba upravit do jednorozměrného vektoru (Full-Connected layer). Tento vektor je pak následně jako vstup do neuronové sítě, která vypočítá konečný výstup ze sítě. Tento vektor se získá takovým způsobem, že se propojí každý neuron s neuronem z předchozí vrstvy. Lze mít také více těchto vrstev v síti. Jako aktivační funkce slouží ReLu, tyto aktivační funkce se rovněž používají i v konvolučních vrstvách. [4, 12]



Obrázek 11: Konvoluční neuronová síť. [11]

4.2 Tvorba konvoluční sítě

- **Konvoluční vrstva (Convolutation layers) -** aplikuje speciální počet filtrů na obrázek. Pro každý subregion obrázku vrstva vypočítá pomocí množiny matematických operací jedinou hodnotu na výstupu. Konvoluční vrstva používá typickou ReLu aktivační funkci, která vkládá do modelu nelinearitu.

- **Škálovací vrstva (Pooling layer) -** snižuje velikost obrázků extrahovaných konvolučními vrstvami, tak aby se zmenšila dimenze mapy a zkrátila čas zpracování. Obvykle se používá max-pooling, který vytváří subregiony mapy s velikostí 2×2 pixelů, udržuje jejich maximální hodnotu a zbavuje se všech ostatních hodnot. Lze ale použít i mean pooling nebo jiný.

- **Plně propojená vrstva (Fully connected layer) -** provádí klasifikaci na extrahovaných datech z konvolučních a škálovacích vrstev. V plně propojené vrstvě je každý uzel vrstvy propojen s každým uzlem z předchozí vrstvy.

Síť je typicky poskládána z více konvolučních modulů, které provádějí extrakci dat. Každý tento modul je tvořen konvoluční vrstvou, po niž následuje vrstva škálovací. Po posledním modulu následuje jedna nebo více plně propojených vrstev, které provádějí klasifikaci. Konečná vrstva obsahuje jediný uzel pro každou klasifikovanou třídu v modelu (všechny možné třídy,

které lze v modelu klasifikovat). Z těchto údajů lze poté zjistit pravděpodobnost, že snímek spadá do dané kategorie např. auto. [20]

4.3 Učení

Při učení pomocí backpropagation se mění synaptické váhy pouze u plně propojovací vrstvy a u konvolučních vrstev se upravují konvoluční jádra (filtry). Toto zaručuje vyšší výkon a menší paměťovou náročnost. Při učení se tvoří hierarchie příznaků a tím se zvyšuje složitost, např. prvně jsou body poté hrany a nakonec tvary. Na podobném principu funguje i lidský mozek.

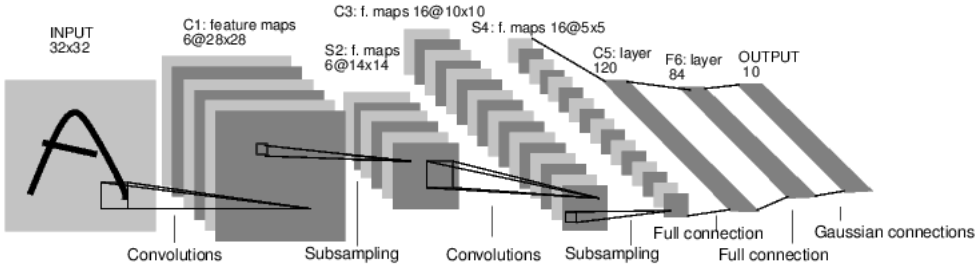
Pro získaní chyby lze využít např. funkci *Softmax* s pomocí křížové entropie (viz kapitola 7.4), kdy je snaha se co nejvíce přiblížit nule.

Plně propojovací vrstva se často doplňuje o funkci *Dropout*, která se využívá k rychlejšímu učení sítě a pro nepřeučení sítě, kdy se ignorují aktivační hodnoty i náhodných neuronů z nerunové sítě při učícím procesu. Vzniká tak umělý šum. Dropout se využívá v konvolučních vrstvách i v plně propojovacích. [28]

4.4 Modely konvoluční sítě

LeNet-5

Typickou a zároveň první konvoluční sítí je model LeNet-5, kterou vytvořil Yannim LeCun se svým týmem v roce 1998, který sloužil pro rozpoznání ručně psaných číslic. Tato síť se skládá ze sedmi vrstev. Její vyobrazení je na obrázku 12. Původně se využívala mean-pooling na místo max-pooling, první max-pooling pak byl použit až v roce 1992 v upraveném modelu Cresceptronu a v modelu HMAX v roce 1999. Tyto modely dosahují nejlepších vyhodnocovacích výsledku na datasetu MNIST. [4, 25, 30]

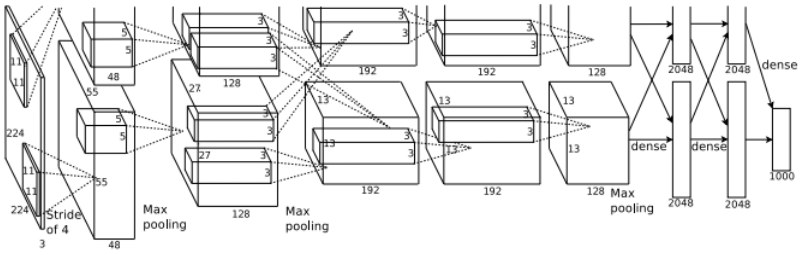


Obrázek 12: Síť LeNet-5. [28]

AlexNet

V roce 2012 překonala síť AlexNet (obrázek 13) předchozí soutěžící v ILSVRC, kdy snížila výrazně svou chybovost oproti konkurenci. Síť měla velmi podobnou topologii jako LeNet, ale

byla s větším počtem filtrů na vrstvě. AlexNet byl trénován simultánně na dvou Nvidia Geforce GTX 580 GPUs. Proto byla síť rozdělena do dvou proudů. Síť byla vytvořena skupinou SuperVision. [28]

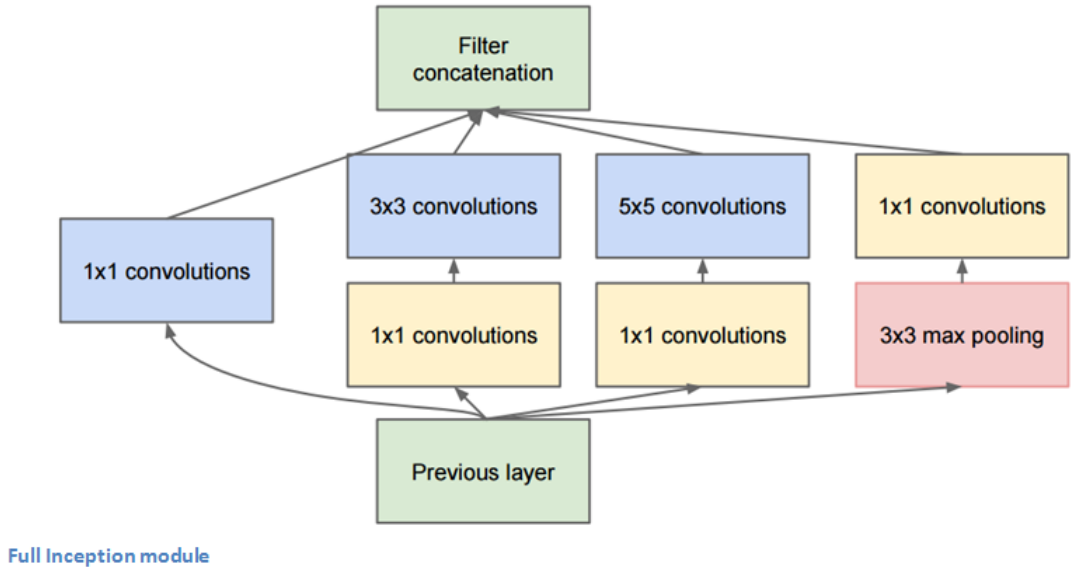


Obrázek 13: Síť AlexNet. [28]

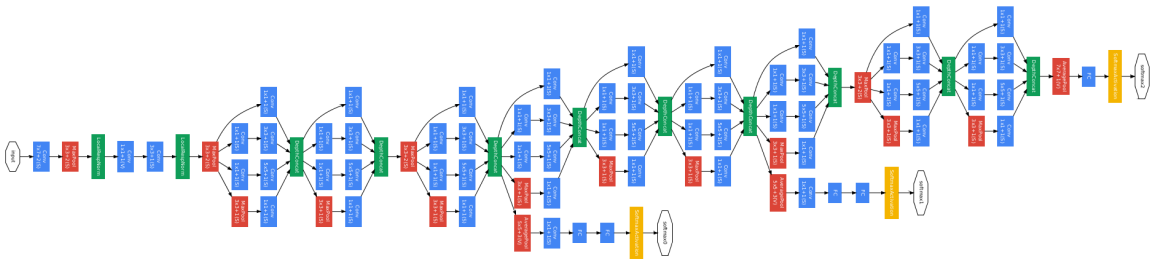
GoogleNet/Inception

V roce 2014 se vítězem na ILSVRC stala síť GoogleNet (obrázek 15) nebo také Inception od Googlu tvořena 22 vrstvami, která měla velice blízko k lidské chybovosti. Chybovost sítě byla pouhých 5.7% oproti AlexNet s 15.4%. Síť byla inspirována sítí LeNet, ale byl do ní implementován nový prvek - Incepční modul, schéma modulu je vyobrazené na obrázku 14. Tento model je založen na několika menších konvolučních vrstvách, aby tak snížil počet parametrů. Byla to první konvoluční síť, která se oprostila od obecné struktury sítě. [30, 31]

Myšlenka Incepční vrstvy se zabývá tím, jak pokrýt co největší oblast, ale zároveň zachovat jemnější rozlišení pro dostatek detailních informacích na obrázku. Toho bylo docíleno incepční vrstvou, která v sobě skloubila souběžnost různě velikých filtrů od malých velikostí pro detaily (1×1) až po velké (5×5) filtry. Síť GoogleNet nemá pouze jeden výstup, ale obsahuje celkem tři výstupy a to po třetí incepční vrstvě, šesté incepční vrstvě a hlavní výstup po sedmé incepční vrstvě, který je zároveň poslední vrstvou celé sítě vyobrazené na obrázku 15. [30]



Obrázek 14: Incepční modul. [31]



Obrázek 15: Síť GoogleNet modul (větší obrázek v příloze C) [32]

5 Vývoj

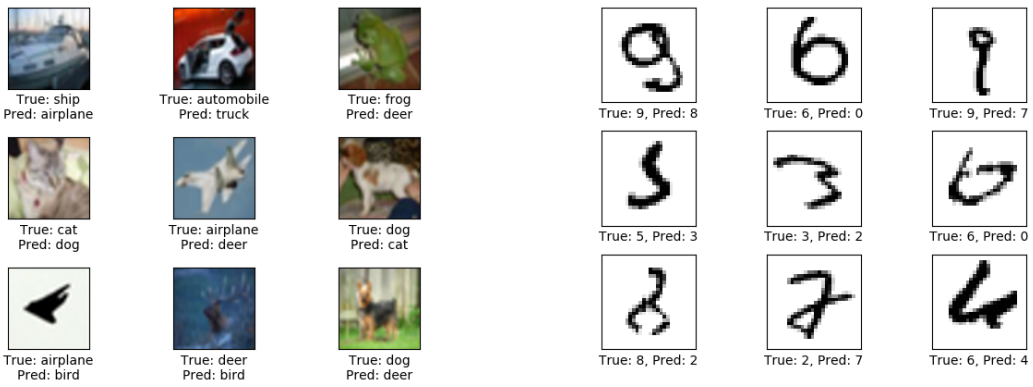
Pro vývoj konvoluční neuronové sítě byl zvolen multiplatformní, interpretovaný, objektově orientovaný programovací jazyk Python. Byl vybrán z toho důvodu, že umožňuje relativně jednoduchý a rychlý vývoj s velkou podporou knihoven a frameworků pro tvorbu neuronových sítí.

Celý vývoj byl absolvován na notebook LENOVO ideapad 710S PLUS s CPU Intel Core i7-7500U 2.7 GHz, 8GB pamětí DDR4 a GPU NVIDIA GeForce 940MX 2GB GDDR5 s nainstalovanou CUDOU verze 8.0.61, cuDnn v.7.0 s operačním systém Linux Ubuntu ve verzi 17.10. Lze tak demonstrovat, že neuronové sítě již nejsou dostupné pouze pro superpočítače, ale lze je také sestrojit na dnes lepší počítačové sestavě.

Výsledný program byl poté implementován za pomocí frameworku TensorFlow (viz kapitola 7), který je vyvíjen Googlem. Byl vybrán kvůli praktické dokumentaci, jednoduchosti, podpoře výpočtu na GPU a široké komunitě vývojářů.

Jako další framework, ve kterém byla síť implementována, byl framework Torch, respektive Pytorch (viz kapitola 8), který podporuje programovací jazyk Python. Opět také podporuje výpočty na GPU, má rozsáhlou dokumentaci na webu a je široce podporován komunitou. V porovnání s TensorFlow bylo dosáhnuto mnohem lepšího výsledku.

Za pomocí obou frameworků byla vyhotovena konvoluční neuronovou síť s modelem GoogLeNet. Otestována na dvou datasetech MNIST [34] - obrázky ručně psaných čísel a CIFAR10 [35] obsahující obrázky roztříděných do deseti kategorií - letadlo, auto, pták, kočka, jelen, pes, žába, kůň, loď, nákladní automobil.



Obrázek 16: Vlevo vyobrazen dataset CIFAR10 [34], vpravo dataset MNIST. [35]

6 Frameworky

Chceme-li vývoj sítě jednodušší a nemuset začínat od nuly, využijeme tzv. frameworky, které řeší obsluhu neuronové sítě. Uživatel si může vše vytvořit sám a nepoužívat frameworky, většinou však sáhne po použití některého frameworku. Mimo úsporu času jsou také optimalizovány jak pro CPU, tak GPU a stále vyvíjeny tak, že sledují aktuální trendy. Pro práci s neuronovými sítěmi existuje nespočet frameworků.

6.1 Caffe

Caffe je deep learningový framework vytvořen s důrazem na rychlost a modularitu. Je vyvíjen Berkeley AI Research (BAIR) a komunitou. Vytvořil jej Yangqing Jia během svého doktorandského studia.

Podporuje práci na CPU i GPU. Caffe dokáže zpracovat přes 60mil obrázků za den na NVIDIA K40 GPU. Je stále aktivně vyvíjen. Jeho hlavní použití je na zpracování obrazu, kde je také nejpoužívanější framework. Je napsán v jazyce C/C++ s interfacem na Python nebo MATLAB. [13]

6.2 TensorFlow

TensorFlow je veřejně dostupný framework pro numerické výpočty s využitím toku datových grafů. Byl vyvinut výzkumnou organizací Machine Intelligence od Googlu. Flexibilní architektura umožňuje nasadit výpočty na jeden nebo více CPU nebo GPU na stolním počítači, serveru nebo mobilním zařízení s použitím jednotného API. Původně byl navržen pro účel výuky strojů a výzkumu neuronových sítí Google Brain týmem. Ale systém se osvědčil a je použitelný v celé řadě domén.Podporuje vývoj v C++ a Pythonu. [14]

6.3 Torch

Torch je vědecký počítačový framework s širokou podporou pro strojové učení s podporou GPU. Je snadno ovladatelný a efektivní díky snadnému a rychlému skriptovacímu jazyku LuaJIT a implementaci C / CUDA.

Cílem Torch je maximální flexibilita a rychlost při vytváření vědeckého algoritmu, přičemž proces je velice jednoduchý. Torch má velkou komunitu, která pro něj připravila nespočet balíčku v oblasti strojového učení, ať už k zpracování obrazu, zvuku nebo zpracování signálu.

Srdcem frameworku jsou neuronové sítě a optimalizační knihovny, které jsou snadno použitelné a mají maximální flexibilitu při implementaci topologie neuronové sítě. Všechno lze jednoduše a efektivně paralelizovat na CPU či GPU. Samotný torch podporuje vývoj v C/C++ nebo Lua. Některé modifikace však podporují i rozhraní Python. Konkrétně např. Pytorch. [15]

6.4 Theano

Theano je Python knihovnou, která umožňuje efektivně definovat, optimalizovat a vyhodnocovat matematické výrazy zahrnující multidimenzionální pole. Umožňuje provádět výsledky za pomocí GPU. Podporuje vývoj pouze v jazyce Python. [36]

6.5 Caffe2

Caffe2 je deep learning framework umožňující jednoduché a flexibilní hluboké učení, který je postavený na původním Caffe. Caffe2 je navrhnut s důrazem na rychlost a modularitu. Což umožňuje flexibilní organizaci výpočtů. Caffe2 má za cíl poskytnout jednoduchý a přímočarý způsob jak experimentovat s hlubokým učením s přispěním široké komunity s pomocí nových modulů a algoritmů.

Umožňuje pracovat s rozhraním Python nebo C++, které je vzájemně zaměnitelné, takže je možné vytvářet prototypy, jež lze rychle později optimalizovat. Od základů je postaveno tak, aby plně spolupracoval s nejnovějšími knihovnami NVIDIA Deep Learning SDK, cuDNN, cuBLAS a NCCL a přinesl tak vysokou akceleraci s více grafickými procesory stolních počítačů, serveru nebo datových center. [16]

6.6 CNTK

Microsoft Cognitive Toolkit (CNTK) je otevřený nástroj pro komerční tvorbu distribuované hluboké učení. Popisuje neuronovou síť jako sérii výpočetních kroků přes orientovaný graf. CNTK umožňuje jednoduchou realizaci a kombinaci oblíbených modelů dopředné neuronové sítě, konvoluční neuronové sítě a rekurentní sítě. Implementuje stochastic gradient sestup s automatickou diferenciací a paralelou na více GPU a serverech. [17]

CNTK lze vložit jako knihovnu v Python, C# nebo C++ programech nebo použít jako samostatný učební nástroj pomocí vlastního popisného jazyka BrainScrip. CNTK je také jeden z prvních nástrojů deep learningu umožňující podporu Open Neural Network Exchange (ONNX) formátu, což je sdílené modelové řešení pro interoperabilitu a optimalizaci. Podporovaný např. Microsoftem, Facebookem a mnoho dalšími společnostmi. ONNX umožňuje implementovat modely napříč frameworky jako např. CNTK, Caffe2 nebo Pytorch. [17]

6.7 MXnet

MXnet je deep learning framework navržen efektivně a zároveň flexibilně, který umožňuje kombinovat symbolické programování a imperativní programování s cílem maximalizovat efektivitu a produktivitu. Je tvořen dvěma vysokoúrovňovými rozhraními a to Gluon API a Module API. [18]

Gluon může vytvořit jednoduchý prototyp, sestavit ho a následně trénovat bez obětování rychlosti trénování tím, že umožňuje intuitivní imperativní vývoj Python kódu a zároveň rychlejší spouštění automatického generování symbolického grafu za použití hybridizace. Proto je pro

začátečníky doporučeno začínat s Gluon API.MXnet podporuje rozhraní Python, R, C++ a Julia. Většinu tutoriálů se však nachzí v Pythonu. [18]

6.8 Chainer

Chainer je flexibilní framework pro neuronové sítě. Jedním z hlavních cílů je flexibilita, tím umožňuje jednoduché a intuitivní psaní komplexních architektur sítě. Většina existujících frameworků je založena na systému "Define and Run". To znamená, že síť je pevně definovaná. Vzhledem k tomu, že je staticky definovaná před jakýmkoliv výpočtem, musí být veškerá logika vložena do datové struktury sítě. [19]

Chainer je však založen na principu "Define by Run". Tzn. že síť je definovaná až před skutečným výpočtem. Jedná se tak o dynamicky definovanou neuronovou síť. V sítí Define and run již není po incializaci možné provádět dynamické změny jako jsou cykly, podmínky, dynamické hodnoty atd. Zatímco Define by run to dovoluje, dokonce má možnost vidět aktuální stavy na vrstvách a podle toho se síť muže během učení měnit. Tyto sítě jsou obecně silnější než statické, daň si pak vybírají na trénování, které je o poznání těžší. [19]

6.9 Keras

I když se nejedná o tradiční framework zajištující práci s neuronovými sítěmi, i tak stojí za zmínku.

Keras je vysokoúrovňová API neuronová síť napsaná v Pythonu, která umožňuje rychlejší a jednodušší návrh neuronové sítě. Je dělán pro lidi a ne pro stroje. Pro svůj běh však potřebuje další framework a TensorFlow, CNTK nebo Theano. Bez jednoho z nich nelze plnohodnotně vytvořit neuronovou síť. Vývoj je možný pouze v jazyce Python. [14, 36]

7 TensorFlow

7.1 Instalace

TensorFlow lze nainstalovat na tři nejčastější operační systémy a to Windows, macOS nebo Linux. Vše je jednoduše sepsáno na webových stránkách. Pokud se jedná o zpracování dat na GPU, je potřeba ještě nainstalovat CUDA toolkit 7 a vyšší, cuDNN v3 nebo vyšší a zároveň musí mít grafická karta podporu CUDA Compute Capability 3.0 a více. Po téhle instalaci TensorFlow bude zpracovávat data na grafické kartě. Celá instalace je velice podrobně popsána i s širokou škálou možností. Framework byl testován na verzi TensorFlow 1.2.0 s podporou GPU.

7.2 Základ

Pro TensorFlow (TF) exituje na webu spoustu tutoriálů. Jsou obsáhlé a základní informace o tvoření neuronové sítě jsou jednoduše sepsány. Jelikož byla vytvářena konvoluční neuronovou síť GoogLeNet, budou dále pouze tyto.

7.3 Tvorba sítě

Základem je model. Je spousta způsobu, jak v TF vytvořit model. Lze použít knihovnu Keras nebo použít xpretty pro návrh modelu. V aplikaci byla však použita nejzákladnější tvorba pomocí vrstev.

Příprava dat

```
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
```

Reshape zajišťuje úpravu dat do matice velikosti obrázku. Vstupní parametry

`[batch_size, image_width, image_height, channels ]`

- **batch_size** - velikost množiny obrázku pro trénování
- **image_width,image_height** - šířka a výška obrázků
- **channels** - Barevné složky obrázku. Pokud je obrázek černobílý, složka je pouze jedna a to černá. Pokud je však barevný, skládá se ze tří složek (červené, zelené, modré) - RGB

Namísto batch_size se zadá hodnota -1, to znamená, že hodnota bude dynamicky vypočtena podle x, když bude $x = 100$, hodnota batch_size bude stejná.

První konvoluční vrstva

Na první konvoluční vrstvu se aplikuje 32 filtrů 5×5 s ReLU aktivační funkcí.

```
conv1 = tf.layers.conv2d( inputs=x_image, filters=32, kernel_size=[5, 5],
    padding="same", activation=tf.nn.relu)
```

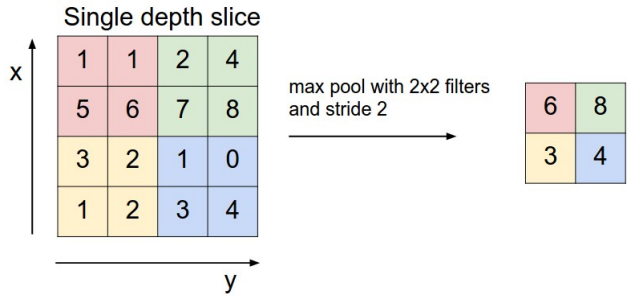
Vstupními parametry jsou *inputs*, což jsou vstupní data. *Filters* - zde se zadává počet filtrů, které budou použity, kernel ve tvaru [šířka, výška], poté *padding*, což je okraj snímku. Zadáním "same"způsobí přidání okrajů s hodnotou 0. Snímek 28×8 na který se nepoužijí okraje, způsobí extrakci pouze z 24×24. Jelikož konvoluční matice nelze vmáčknout na první pozici. Proto se okraje doplní o 0 a neztratíme tím hodnoty. Parametr *activation* určuje aktivační funkci konvoluční vrstvy.

Škálovací vrstva

Po konvoluční vrstvě se tvoří vrstva škálovací. K tomu lze použít funkci *max_pooling2d()*

```
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
```

Parametr *inputs* opět slouží pro vstupní data z vrstvy. Další parametry *pool_size* je specifická velikost max pooling filtru ve formátu [šířka, výška]. Pokud jsou obě velikosti stejné, stačí napsat zkráceně *pool_size* = 2. *Stides* určuje velikost kroků pro extrahování subregionu filtrem. Když se nastaví krok 2 a *pool_size* 2×2, docílí se toho, že se žádná z oblastí nebude překrývat. Lze jí taky zadat ve formátu [šířka, výška].



Obrázek 17: Ukázka škálování pomocí Max poolingu. [21]

Kombinací těmito dvěma vrstvami se vytvoří model. Z poslední vrstvy pak vchází do plně propojovací vrstvy, která klasifikuje třídu obrázku.

Plně propojovací vrstva

Plně propojovací vrstva (Full-connected layer) klasifikuje objekty do daných tříd. Ještě před vstupem je však potřeba výstup z pooling vrstvy upravit, aby byl tensor pouze jednorozměrný.

```
pool_flat = tf.reshape(pool2, [-1, 14 * 14 * 32])
//nebo
pool_flat = tf.contrib.layers.flatten(pool1)
```

Opět reshape -1 znamená dynamicky vypočteno na základě vstupních dat. Čtrnáct je pak šířka a výška obrázků a 32 počet filtru. Druhý příklad si dále vše řeší sám na základě vstupních dat. Teď lze použít plně propojovací vrstvu.

```
dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.relu)
```

Inputs parametr musí být ve specifickém tvaru "Flat". Počet neuronů v plně propojovací vrstvě určuje *units* a *activation* zajišťuje aktivační funkci neuronů.

Pro zlepšení výsledku během tréninku lze použít funkci dropout, která specifikuje míru výpadku. Použitím 0.4, bude 40% náhodných prvků vynecháno. Tato funkce se používá pouze při tréninku.

```
dropout = tf.layers.dropout(inputs=dense, rate=0.4, training=mode == tf.
    estimator.ModeKeys.TRAIN)
```

Poslední vrstvou je vrstva klasifikační. Tvoří jí plně propojovací vrstva s počtem neuronů podle počtu tříd, do který lze zařadit prvky.

```
logits = tf.layers.dense(inputs=dense, ,units=num_classes, activation=None)
y_pred = tf.nn.softmax(logits=logits)
y_pred_cls = tf.argmax(y_pred, dimension=1)
```

Jelikož jsou výstupy ze plně propojovací vrstvy příliš nízké nebo vysoké, je třeba je normalizovat např. pomocí Softmax funkce, jejíž výstupem jsou hodnoty v rozmezí 0 až 1. Zatímco funkce argmax vrací index nejvyšší hodnoty pravděpodobnosti a z toho se odvodí třída do které spadá.

7.4 Trénování

Aby model lepé klasifikoval vstupní obrázky, musí se naučit změnit proměnné (variables) pro všechny síťové vrstvy. Aby se tak stalo, musí se zjistit, jak dobře funguje model teď porovnáním předpokládaného výstupu *y_pred* a požadovaného výstupu *y_true*.

Cross-entropy je měřítko výkonnosti použitého při klasifikaci. Je to spojitá funkce, která je vždy kladná. Pokud se výstup rovná s předpokládaným výstupem je roven nule. Cílem je tedy optimalizovat cross-entropy, tak aby se blížila nule. Toho se docílí změnou proměnných na síťových vrstvách.

TensorFlow má pro výpočet cross-entropy vestavěnou vlastní funkci. Funkce interně vypočítává softmax, proto na vstupu nesmí být hodnoty vystupující ze softmax funkce, ale přímý výstup z poslední plně propojené vrstvy *y_pred*.

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=
    logits)
```

Vypočtením cross-entropy pro každou klasifikaci obrázku se zjistí, jak efektivně síť pracuje. Ale aby se mohlo použít cross-entropy pro optimalizaci sítě, je zapotřebí použít skalární hodnotu, což je průměr cross-entropy pro všechny klasifikace.

```
loss = tf.reduce_mean(cross_entropy)
```

Optimalizace

Hodnota *loss*, by se měla co nejvíce blížit nule. Díky loss lze vytvořit optimalizátor. Např. AdamOptimalizer, který je pokročilou formou gradientového sestupu.

```
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
```

V tomto okamžiku optimalizátor nic nepočítá, je pouze připraven na pozdější použití. Pro pokrok bude potřeba více informací.

```
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
```

Vektor booleanu, pokud se předpokládaný výstup rovná výstupu.

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Vypočítá přesnost za základě *correct_prediction*, převede vše na float tzn. True = 1, False = 0 a spočítá jejich průměr.

Jestliže byl graf sestaven, je třeba jej spustit a inicializovat proměnné (**weights a biasses**) před tím než budou optimalizovány.

```
session = tf.Session()
session.run(tf.global_variables_initializer())
```

Nyní byl sestaven model a pomocné proměnné pro práci se sítí. Tomuto se říká graf. Pokud je graf hotový stačí již pouze vlastní funkce pro trénink nebo zobrazení informací.

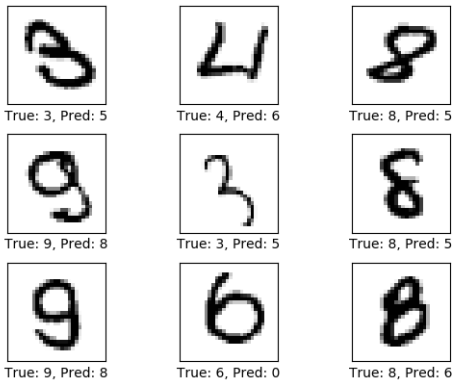
7.5 Práce s datasetem

Neuronová síť byla vyvíjena pro dva data-sety a to MNIST obrázek 16, což jsou ručně psaná čísla a CIFAR10 obrázek 16, který obsahuje 10 kategorií (letadlo, auto , pták, kočka, jelen, pes, žába, kůň, loď, nákladní auto).

V určitých věcech se data-sety liší, proto bylo třeba přistupovat k data-setům rozdílně. Když se jednalo například o vykreslovaní či práci s daty.

TensorFlow již obsahuje implementovanou funkci na stažení datasetu MNIST, proto i práce s datasetem byla o poznání snazší. Zatímco u CIFAR10 bylo třeba použit externí knihovnu CIFAR10, která je volně k použití a slouží ke stažení a před připravení dat z datasetu.

K vykreslení jednotlivých obrázků byla použita knihovna matplotlib.pyplot, která se implementuje ve funkci plot_images. Jejími parametry jsou data obrázků, jejich kategorie a pole predikce, které se zobrazí pouze tehdy, pokud tento parametr použijeme. Funkce zpracuje vstupní parametry a vykresli 9 obrázku. Je nutno, aby vstupní pole bylo velikosti 9 a v matici 3×3. S informací do jaké kategorie patří "True", popřípadě jaké kategorii byla předpovězena "Pred". Ukázkový výpis na obrázku 18.



Obrázek 18: Ukázka zobrazení obrázků v 3x3 matici.

7.6 Model

Úkolem bylo navrhnou síť GoogLeNet, která je tvořena incepčními vrstvami navazujícími mezi sebou propojeními. Model byl navrhován pomocí tf.layers. Lze však síť tvořit i pomocí frameworku Keras nebo jiného frameworku pro návrh modelu sítě.

Model je složen z incepčních vrstev tvořené pomocí funkce *inception_layer* jejími parametry: *inputs*, což jsou vstupní data do vrstvy. Poté parametry $cov\_[Velikostivrstvy]$ a $pool\_size$, které určují počet filtrů pro jednotlivé vrstvy, které odpovídají počtu výstupních kanálů a nakonec název vrstvy. Z funkce vychází jako výstup hotová inceptční vrstva. Z těchto jednotlivých vrstev se pak pomocí jejich propojení vytvoří model sítě GoogLeNet. Výstupem funkce *GoogLeNet* je pak poslední zpracovaná vrstva s deseti konečnými uzly, která klasifikuje pravděpodobnost, do které kategorie patří.

7.7 Trénování

Připravený graf je třeba trénovat. Trénování se provádí na datech train. Tímto setem se síť naučí klasifikovat obrázky do tříd pomoci upravením jejich proměnných. Aby bylo jasné, jak je

síť úspěšná. Síť se testuje na datech test z data-setu. S těmito obrázky nepřišla síť do styku při trénování.

Funkce *optimalize* přijímá parametr num_iterations, který udává počet iterací učení. V programu existuje i funkce Optimalize_Infinity, který funguje v nekonečném cyklu a trénuje tak síť do nekonečna. V každé iteraci načte do proměnné feed_dict_train náhodnou sadu obrázku a k nim příslušné rozřazení do kategorií. Tato proměnná pak vstupuje do grafu společně s optimalizérem, který podle výsledku upraví proměnné v síti. Po každých 100 iteracích pak bude vypsána informace o přesnosti predikce.

Pokud je přesnost u testovacích dat vyšší než předešlá nejvyšší úspěšnost, které byla dosažena, budou uloženy hodnoty proměnných. Díky tomu, pak lze síť přerušit a nepřijít o vytrénovaná data. Po ukončení trénování pak bude spočítána úspěšnost a výpis délky trvání tréninku. Bude-li nutné, lze vykreslit špatně vyhodnocené obrázky nebo konfúzní matici zobrazující správnost a chybovost algoritmu.

Pro výpočet úspěšnosti v procentech používám vzorec:

$$accuracy = \frac{correct\_sum}{images} \cdot 100 \tag{6}$$

Kde:

- **accuracy** = přesnost v %
- **correct_sum** = počet úspěšně předpovězených
- **images** = počet vyhodnocujících obrázku

```
Iterace:    100, Train-Batch Accuracy:  96.9%, Validation Acc:  96.7%  - Save
Iterace:    200, Train-Batch Accuracy:  95.3%, Validation Acc:  95.3%
Iterace:    300, Train-Batch Accuracy:  96.9%, Validation Acc:  96.7%
Iterace:    400, Train-Batch Accuracy:  96.9%, Validation Acc:  97.3%  - Save
Iterace:    500, Train-Batch Accuracy:  95.3%, Validation Acc:  96.1%
Iterace:    600, Train-Batch Accuracy:  96.9%, Validation Acc:  96.9%
Iterace:    700, Train-Batch Accuracy:  96.9%, Validation Acc:  97.2%
Iterace:    800, Train-Batch Accuracy:  98.4%, Validation Acc:  97.8%  - Save
Iterace:    900, Train-Batch Accuracy:  98.4%, Validation Acc:  97.6%
Iterace:   1000, Train-Batch Accuracy:  95.3%, Validation Acc:  96.6%
Time usage: 0:02:38
Accuracy on Test-Set: 96.4% (9639 / 10000)
```

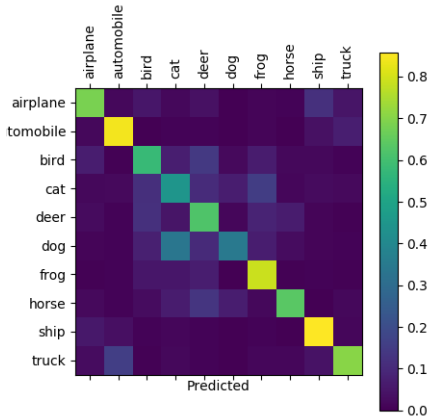
Obrázek 19: Ukázka výpisu během trénování.

7.8 Informační výpisy

Pro větší přehled o přesnosti bylo vytvořeno pár funkcí, které pomohou lépe zobrazit vyhodnocování. Kromě accuracy info zobrazující informace(obrázek 19) o hodnotách sítě, také zobrazení konfúzní matice. Konfúzní matice umožňuje zjistit, které obrazce se špatně klasifikovaly a za co se nejčastěji zaměňují. Je zobrazována, jak matice v terminálovém okně, tak s grafickou podobou, která je na obrázku 20. Tu opět umožňuje knihovna matplotlib.pyplot, která z matice vytvoří barevný graf obrázek 20, který je však oproti konzolové matici normalizovaný.

$$\begin{pmatrix}
0.687 & 0.019 & 0.052 & 0.017 & 0.039 & 0.000 & 0.012 & 0.009 & 0.011 & 0.047 \\
0.020 & 0.842 & 0.001 & 0.008 & 0.008 & 0.000 & 0.011 & 0.002 & 0.037 & 0.071 \\
0.070 & 0.006 & 0.578 & 0.072 & 0.141 & 0.022 & 0.066 & 0.017 & 0.019 & 0.009 \\
0.019 & 0.022 & 0.115 & 0.449 & 0.107 & 0.070 & 0.157 & 0.015 & 0.025 & 0.021 \\
0.025 & 0.007 & 0.118 & 0.048 & 0.0622 & 0.015 & 0.084 & 0.066 & 0.011 & 0.004 \\
0.011 & 0.007 & 0.075 & 0.338 & 0.103 & 0.346 & 0.70 & 0.028 & 0.013 & 0.009 \\
0.004 & 0.010 & 0.053 & 0.053 & 0.068 & 0.002 & 0.789 & 0.005 & 0.010 & 0.006 \\
0.023 & 0.008 & 0.027 & 0.064 & 0.132 & 0.068 & 0.019 & 0.640 & 0.002 & 0.017 \\
0.054 & 0.034 & 0.008 & 0.016 & 0.007 & 0.000 & 0.008 & 00.00 & 0.858 & 0.15 \\
0.028 & 0.158 & 0.006 & 0.019 & 0.010 & 0.001 & 0.015 & 0.015 & 0.043 & 0.705
\end{pmatrix}$$

Konfúzni matice z klasifikace CIFAR10 datasetu.



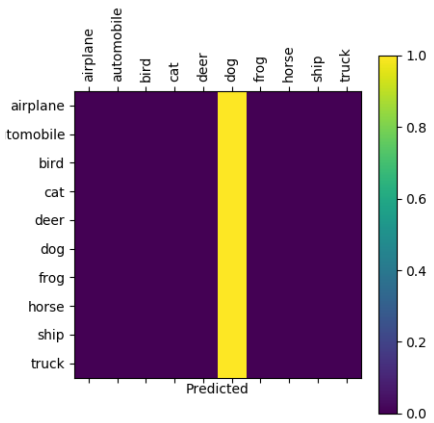
Obrázek 20: Konfúzní matice vykreslená do grafu.

7.9 Vyhodnocení

Pomocí neuronové sítě lze docílit relativně vysoké přesnosti rozpoznání obrazu. Aby však byla vysoká přesnost predikce je třeba, aby se síť trénovala. Jak již bylo dříve zmíněno, není nutné mít na tyto výpočty superpočítač. Lze to provést na lepším osobním počítači s grafickou kartou Nvidia.

Dobrých výsledků lze docílit i po méně iteracích. V následující kapitole se nachází výsledky trénování a postupné rozdíly, ke kterým docházelo díky delšímu trénování. Výsledky vyhodnocování byly testovány na datasetu CIFAR10, jelikož dataset MNIST je velice jednoduchý na naučení a po 5 minutách trénování se dostáváme na 96% úspěšnost rozpoznání.

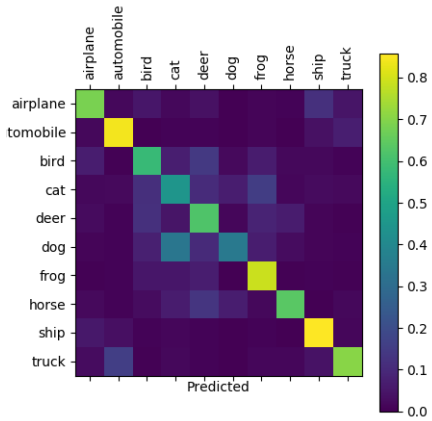
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Výsledek 1: Konfúzní matice za datasetu CIFAR10 s přesností 10%.

Ve výsledku jedna neproběhlo žádné trénování. Váhy byly náhodně vloženy a pro klasifikaci je sít nepoužitelná.

$$\begin{pmatrix} 0.687 & 0.019 & 0.052 & 0.017 & 0.039 & 0.000 & 0.012 & 0.009 & 0.011 & 0.047 \\ 0.020 & 0.842 & 0.001 & 0.008 & 0.008 & 0.000 & 0.011 & 0.002 & 0.037 & 0.071 \\ 0.070 & 0.006 & 0.578 & 0.072 & 0.141 & 0.022 & 0.066 & 0.017 & 0.019 & 0.009 \\ 0.019 & 0.022 & 0.115 & 0.449 & 0.107 & 0.070 & 0.157 & 0.015 & 0.025 & 0.021 \\ 0.025 & 0.007 & 0.118 & 0.048 & 0.0622 & 0.015 & 0.084 & 0.066 & 0.011 & 0.004 \\ 0.011 & 0.007 & 0.075 & 0.338 & 0.103 & 0.346 & 0.70 & 0.028 & 0.013 & 0.009 \\ 0.004 & 0.010 & 0.053 & 0.053 & 0.068 & 0.002 & 0.789 & 0.005 & 0.010 & 0.006 \\ 0.023 & 0.008 & 0.027 & 0.064 & 0.132 & 0.068 & 0.019 & 0.640 & 0.002 & 0.017 \\ 0.054 & 0.034 & 0.008 & 0.016 & 0.007 & 0.000 & 0.008 & 00.00 & 0.858 & 0.15 \\ 0.028 & 0.158 & 0.006 & 0.019 & 0.010 & 0.001 & 0.015 & 0.015 & 0.043 & 0.705 \end{pmatrix}$$



Výsledek 2: Konfúzní matice za datasetu CIFAR10 trénovaný 10000 iteracemi s přesností 62.8%.

Z výsledku 2 bylo správně určeno 6286 obrázku z 10 000. Trénování trvalo cca 37 minut a proběhlo přibližně 10000 iterací trénování. Z výsledku jde vyčíst, že nejčastěji si síť zaměnila kočku za psa a naopak. Lze vidět, že výsledky nejsou vůbec špatné a už po 30 minutách trénování vznikla uspokojivá predikce.

8 Pytorch

Torch lze nainstalovat pouze na systémy s Unixovým jádrem tzn. MacOs a Linux. Samotný torch nepodporuje implementace v Python, proto jsem pro aplikaci použil Pytorch, který staví na knihovně Torch a umožňuje implementaci v jazyku Python. Pytorch je tvořen s důrazem na využití výkonu GPU, flexibilitu a rychlost.

Pytorch má jedinečný způsob budování neuronových sít. Většina frameworku jako TF, Theano, Caffe, CNTK mají statický pohled. Pytorch používá speciální techniku nazvanou Reverse-mode auto-differentiation, která umožňuje měnit síť libovolně s nulovým zpožděním, vycházející z Chainer a jemu podobných frameworku, je to tedy dynamická neuronová síť. Technika sama o sobě není unikátní, ale jedná se o nejrychlejší implementaci doposud. [22]

Reverse-mode auto-differentiation je speciální způsob zpracování derivací, který se používá při n vstupech $x_1, x_2, ..., x_n$, které vytváří jeden výstup x_N. A je potřeba zjisti deriváty funkce $\frac{dx_N}{dx_i}$ pro všechny i. [33]

Práce s Pytorch je intuitivní a lineární v myšlení. Při ladění jsou chyby přesně definovány a lze je přímočaře pochopit bez dlouhého hledání. [22]

8.1 Instalace

Na web stránkách Pytorch se nachází podrobný postup pro instalace frameworku. Pokud vyvíjíme síť i na GPU, je třeba mít stažené knihovny pro práci s CUDA. Problém může nastat, když nejnovější verze Pytorch nebude podporovat grafickou kartu. Proto před instalací je potřeba zjistit, které grafické karty verze podporuje. Stačí poté nainstalovat danou verzi a problém nevznikne. Pokud bude nainstalována špatná verze, chytrá hláška napoví, že GPU je již zastaralé a je třeba buď obměnit grafickou kartu nebo vyměnit za starší verzi Pytorch.

Na webových stránkách se nachází rozsáhlý tutoriál i s ukázkovými příklady. K práci byl použit ve verzi 0.3.0 s podporou CUDA 8.

8.2 Tvorba sítě

K základnímu prvku sítě opět patří model. Model si lze vytvořit samostatně nebo lze použít předdefinované modely např. RNN model sítě. Já jsem si však svůj model vytvořil od základu sám.

Tvorba modelu

Model se tvoří jako potomek třídy *torch.nn.Module*. Obsahuje samotné vrstvy i funkci **forward** ve které jsou vrstvy pospojovány a výstupem je pak vrstva s výsledkem predikce.

Tvorba vrstev

Vrstvy se tvoří pomocí funkci z knihovny *torch.nn*. Conv2D tvoří základní kámen tvorby neuronové sítě.

```
conv2D = nn.Conv2d(in_planes, n1x1, kernel_size=1),
```

Obsahuje tři základní parametry. **in_channles** = vstupní velikost kanálu, **out_channels** = výstupní velikost kanálu, což je totožné s filters v sítích TF a *kernel_size*, zde se může zadat ve tvaru [šířka, výška] nebo pouze jako jedno číslo čímž se počítá že šířka a výška jsou totožné. Není zde žádná aktivační funkce, jelikož se konvoluční vrstvy tvoří odlišně než v TF. Tvoří se sekvenci vrstev *(nn.Sequential)* a příkazy jejíž výsledkem je pak výsledná vrstva.

```
self.b1 = nn.Sequential(
        nn.MaxPool2d(3, stride=1, padding=1),
           nn.Conv2d(in_planes, pool_planes, kernel_size=1),
           nn.BatchNorm2d(pool_planes),
           nn.ReLU(True),
       )
```

Pro zlepšení učení se může použít například *nn.BachNorm2D*, jeho vstupem je pak velikost normalizovaného kanálu. Aby se mohla klasifikovat, je třeba mít aktivační funkci. Tu nám zajišťuje vrstva*nn.ReLU*, která pomocí aktivační funkce vyhodnotí data místo aktivační funkce přímo v konvoluční vrstvě.

Jako v TF i tady je škálovací vrstva *nn.MaxPool2d*, která snižuje velikost zpracovávaných dat. Jejími parametry jsou *kernel_size*, který určuje velikost matice, jenž se může zadat ve tvaru [šířka, výška] nebo pouze jednociferně, kdy se počítá, že výška i šířka jsou stejné. Dalším parametrem je *stride*, který určuje krok posunutí masky pro extrahování subregionu a *paddingm*, který přidává okraje vyplněný nulami o určitou délku (viz obr. 9).

Celou tuto sekvenci vrstev spouští funkce *nn.Sequential*, jejíž výstupem je výsledná vrstva, s kterou se dále pracuje. [22]

Vrstvy lze poté ve funkci forward spojovat a upravovat, tuto funkci obsahuje třída modelu. Při skládání je třeba myslet na to, že výstupem z celého modelu musí být vrstva plně propojovací. V Pytorch jí zajišťuje funkce *[vrstva].view*, která spojí uzly a vytvoří jednorozměrný vektor, který se klasifikuje. Pro klasifikaci výsledku se poté používá vrstva *nn.linear* s parametry vstupní a výstupní velikosti. Výstupem by pak měla být vrstva obsahující uzly s klasifikací.

Výstup je nutno dále upravovat. Stejně jako v TF i zde je funkce *nn.Dropout*, která při trénování vynechá určité procento výsledku podle zadaného parametru. Pro vyhodnocení lze na výstupní vrstvu aplikovat *nn.Softmax* funkci, která výsledky normalizuje mezi rozmezí 0 až 1 a poté aplikovat *torch.max*, která vrací index s kategorie datasetu s největší pravděpodobností.

8.3 Trénování

Stejně jako u jiných sítí, i v Pytorch je třeba síť trénovat. Pro správné trénovaní je zapotřebí znát aktuální úspěšnost sítě. Tu lze zjistit pomocí *Cross-entropy*, kterou má Pytorch implementovanou přímo v sobě a již zajišťuje jak cross-etnropy, tak i její skalární hodnotu. Díky křížové entropii lze síti optimalizovat její váhy, k tomu slouží optimalizér. Ten určuje, jak se váhy v síti upraví, jeho parametry je učební konstanta a proměnné k optimalizaci.

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cnn.parameters(), lr=learning_rate)
```

Sestavení modelu samo o sobě síť nemění. Je třeba vytvořit vlastní funkce, které budou model obsluhovat a tím bude docházet k jeho učení.

8.4 Práce s datasetem

Stejně jako TF byla aplikace v Pytorch implementována pro dva datasety. Těmi byl opět dataset Mnist a Dataset CIFAR10 (obrázek 16). Pytorch implementuje v sobě funkce pro přípravu těchto dat a proto jich bylo i v aplikaci využito. K přístupu k datasetům bylo využita funkce import *dset* z knihovny *torchvision*, která umožňuje již v základu i různě transformace obrázku např. otočení. Protože data se využívají po určitých částech, nikdy ne všechny najednou. Implementoval do sebe Pytorch i jakousi funkci, který vybírá vždy sadu obrázku *torch.utils.data.DataLoader* jejíž parametry jsou velikost sady, odkaz na data set a *shuffle*, který přináší přeházenost do datasetu. Dochází tedy k promíchání datasetu.

Výpis datasetu je využíván stejnou funkci jako v implementaci TF (obrázek 18) s jedinou úpravou, a to před vložením dat bylo třeba upravit formát datasetu pomocí knihovny *Numpy*, který upraví dataset do pole.

8.5 Model

Jak již bylo zmíněno v kapitole 4.4, model GoogleNet, který byl implementován, je tvořen incepčmími moduly. Pro tyto moduly byla vytvořena třída *Inception*, která vytváří incepční vrstvy modelu. Její konstruktor obsahuje počet filtrů, které se budou aplikovat v tomto pořadí: (in_nplanes, n1x1, n3x3red, n3x3, n5x5red, n5x5, pool_planes). Tyto moduly jsou poté mezi sebou propojeny až na poslední plně propojovací vrstvě, která je zároveň i vrstva klasifikační. Tímto způsobem je tvořen použitý model GoogleNet.

8.6 Trénovaní

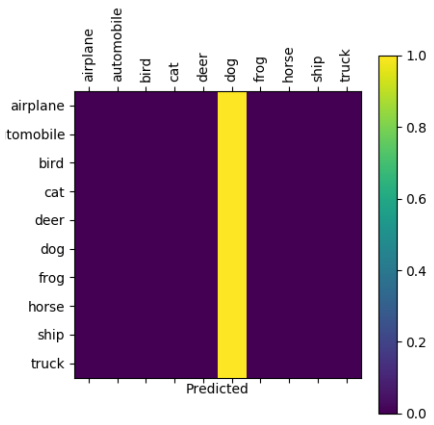
Aby bylo možné síť správně optimalizovat, bylo třeba vytvořit funkce, které budou síť trénovat. Funkce *Optimalize* nebo *Optimalize_infinity* trénují aplikovanou síť. Obě funkce jsou tvořeny smyčkou. *Optimalize_infinity* je tvořen nekonečnou smyčkou, která optimalizuje donekonečna,

zatímco *Optimalize* přijímá parametr, který určuje kolik bude probíhat iterací učení. Samotná funkce pak obsahuje náhodnou sadu obrázku a jejich klasifikaci z datasetu train. Tyto data jsou úpravné jako proměnné a následně vloženy do implementovaného modelu. Ten vrací *output*, který obsahuje predikci do daných kategorii. Tu zpracuje *criterion*, do kterého vstupuje výstup z modelu a pole obsahující správné klasifikace vstupních obrázků. Funkce vrací *loss*, což je skalární hodnota cross-entropy, která by se měla co nejvíce blížit nule. Pomocí těchto dat se síť poté optimalizuje. Informace o výpisu úspěšnosti v procentech je totožná s výpočtem ve vzorci 6 stejně tak výpis konfuzní matici (obrázek 20)nebo výpis v terminálovém okně (obrázek 19).

8.7 Vyhodnocení

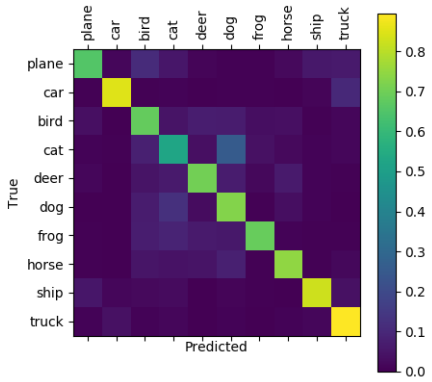
Pro vyhodnocení úspěšnosti byl vybrán dataset CIFAR10, jelikož v datasetu MNIST byl již po menším počtu iterací velice úspěšný a srovnání by nemělo smysl.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Výsledek 1: Konfúzní matice z datasetu CIFAR10 s bez trénování, úspěšnost 10%.

$$\begin{pmatrix}
0.656 & 0.017 & 0.110 & 0.055 & 0.011 & 0.005 & 0.006 & 0.021 & 0.056 & 0.063 \\
0.005 & 851 & 0.006 & 0.010 & 0.003 & 0.006 & 0.004 & 0.002 & 0.011 & 0.102 \\
0.036 & 0.001 & 0.685 & 0.046 & 0.073 & 0.067 & 0.034 & 0.038 & 0.004 & 0.016 \\
0.009 & 0.005 & 0.079 & 0.526 & 0.036 & 0.258 & 0.039 & 0.024 & 0.009 & 0.015 \\
0.017 & 0.003 & 0.048 & 0.063 & 0.706 & 0.067 & 0.018 & 0.064 & 0.009 & 0.005 \\
0.002 & 0.002 & 0.069 & 0.0123 & 0.028 & 0.0728 & 0.006 & 0.033 & 0.007 & 0.002 \\
0.006 & 0.003 & 0.073 & 0.088 & 0.063 & 0.059 & 0.686 & 0.010 & 0.006 & 0.006 \\
0.004 & 0.0 & 0.052 & 0.044 & 0.047 & 0.078 & 0.005 & 0.0746 & 0.005 & 0.019 \\
0.054 & 0.014 & 0.022 & 0.026 & 0.001 & 0.012 & 0.004 & 0.001 & 0.827 & 0.039 \\
0.008 & 0.039 & 0.007 & 0.016 & 0.003 & 0.007 & 0.0 & 0.009 & 0.016 & 0.895
\end{pmatrix}$$



Výsledek 1: Konfúzní matice z datasetu CIFAR10 trenovánáá 10000 iteracemi, úspěšnost 73.0%.

9 Testování

V tabulce 4 lze vidět srovnání TF a Pytorch, kde jsou srovnány data obou frameworků, které byly 10× spuštěny se stejnými parametry. Bohužel Pytorch byl mnohem náročnější na paměť a z tohoto důvodu bylo možno testovat pouze po 8 snímcích v dávce, zatímco v TF bylo možné trénovat s 64 obrázky v dávce. Testování probíhalo po 500 iterací u každého testovaní. Jde vidět, že síť TF je mnohem rychlejší přibližně 9×. I když je Pytorch pomalejší, jedno testování trvá průměrné 13 min a 41 sec u TF je to pouze 01 min a 57 sec, taktéž pracuje s 8× méně daty oproti TF, z krátkodobého hlediska jsou oba frameworky přibližně stejně úspěšné ve vyhodnocení.

Dále bylo prováděno dlouhodobější trénování sítě, při čemž na každé síti proběhlo 20000 iterací. Ukázkové data v tabulce 3. Framework TensorFlow dokázal zpracovat 20000 iterací za 1 hod, 5 min a 55 sec, přičemž dokázal rozpoznat celkem 68.2% obrázků z testovacího datasetu. Zatímco framework Pytorch zpracovával data 9 hod a 40 sec, ve výsledku dokázal rozpoznat data s 77.10% úspěšností.

Tabulka 2: Srovnání frameworků. Celá tabulka v příloze D.

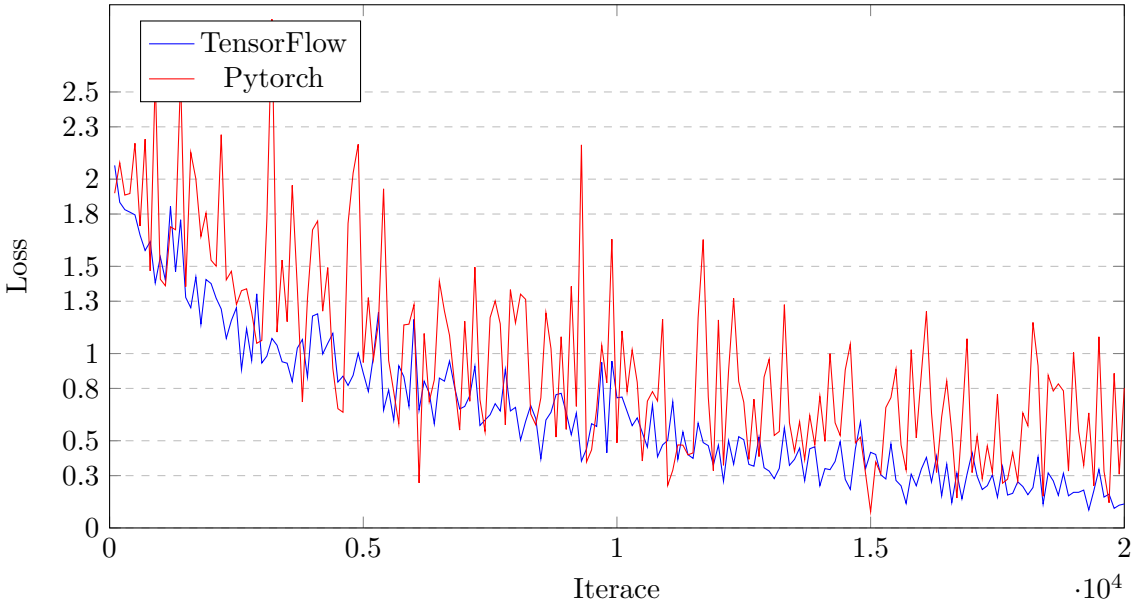
	PČ v s	**PHL**	**PV**	**CČU**	**KVU**
1. testování					
TensorFlow	0.140133	1.883735	25.92%	00:01:55	32.20%
Pytorch	0.544530	1.824981	24.73%	00:14:17	28.73%
2. testování					
TensorFlow	0.140062	1.843850	25,20%	00:01:55	31.50%
Pytorch	0.544195	1.970783	25.40%	00:13:53	26.89%
3. testování					
TensorFlow	0.141323	1.851040	26.60%	00:01:53	29.50%
Pytorch	0.537192	2.117946	19.40%	00:13:35	19.70%
4. testování					
TensorFlow	0.140332	1.893062	24,93%	00:01:56	32,50%
Pytorch	0.538019	2.113416	27.23%	00:13:34	30.43%
5. testování					
TensorFlow	0.139996	1.959011	22.90%	00:01:56	26.60%
Pytorch	0.533269	1.905482	26.80%	00:13:33	32.83%
Pruměrné výsledky ze všech testování					
TensorFlow	0.144015	1.890437	24.97%	00:01:57	30.41%
Pytorch	0.537412	1.969147	24.33%	00:13:41	27.23%

PČ v s - průměrný čas v sekundách,
PHL - průměrná hodnota loss,
PV - průměrný výsledek,
CČU - Celkový čas učení,
KVU - Konečný výsledek učení.

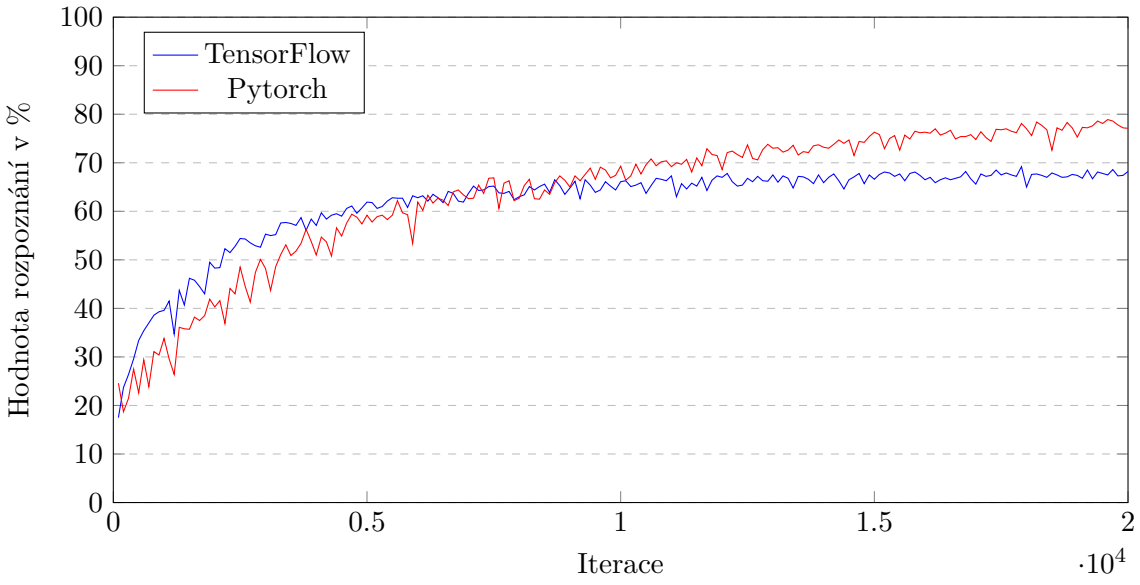
V grafu 21 je srovnání dat loss při učení, kde je znázorněna klesající funkce loss, která by se měla blížit v ideální případě nule. TensorFlow má hodnotu učení 0.0001, zatímco má framework

učební hodnotu 0.001, což zapříčiňuje větší rozptyl v grafu 21. Vyšší hodnota učení zaručuje rychlejší učení zapříčiněno horšími rozpoznávacími výsledky.

V grafu 22 je znázorněné rostoucí učení po každé iteraci v obou frameworcích, kde TensorFlow dosáhl maximálního rozpoznání v iteraci 15000 s 67.8%. Pytorch dosáhl svého nejvyššího rozpoznání v 19600 s 78.9%.



Graf 21: Hodnota loss při učení.



Graf 22: Rozpoznání datasetu CIFAR10.

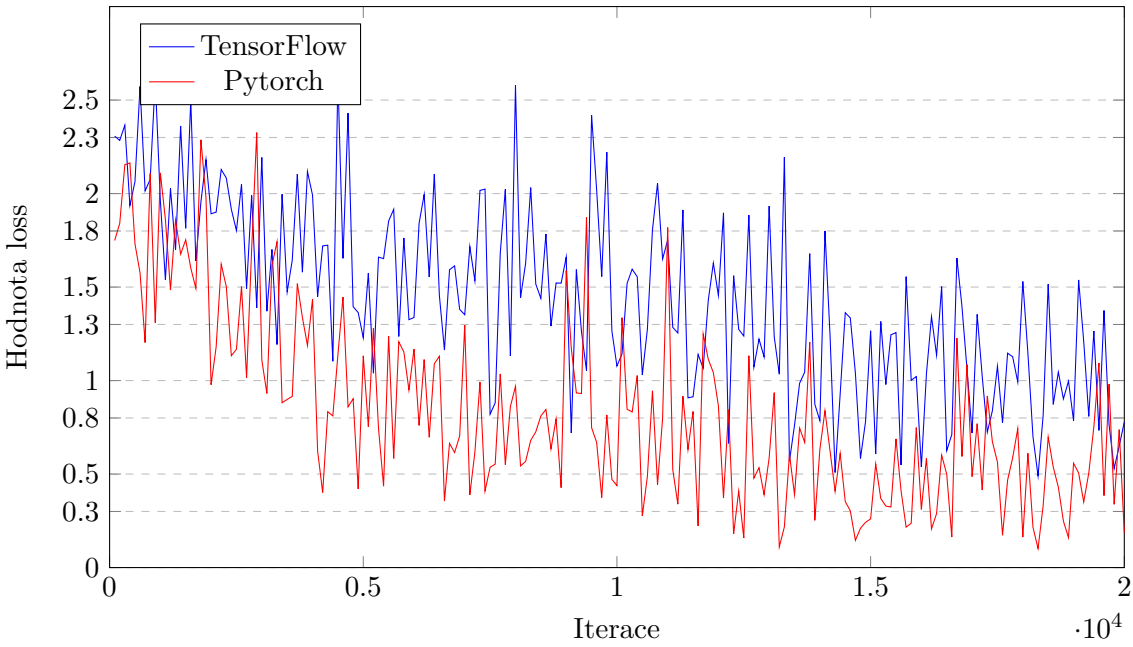
Tabulka 3: Ukázka dat během učení.

Iterace	**ČI v ms**	**HL**	**KVU**
TensorFlow			
100	0.1397600	2.0788707	17.5%
1000	0.1401848	1.5542024	39.6%
5000	0.1401810	0.8879294	61.9%
10000	0.1401870	0.7461596	66.1%
15000	0.1397778	0.4336419	66.6%
17900*	**0.139868**	**0.264723**	**69.2%**
20000	0.1403019	0.1374788	68.2%
Pytorch			
100	0.5340280	1.9193575	24.6%
1000	0.5358498	1.4260094	33.8%
5000	0.5336949	0.9490251	59.2%
10000	0.5358731	0.4880162	69.3%
15000	0.5325760	0.0928165	76.3%
19600*	**0.536979**	**0.325100**	**78.9%**
20000	0.5315229	0.80141174	77.1%

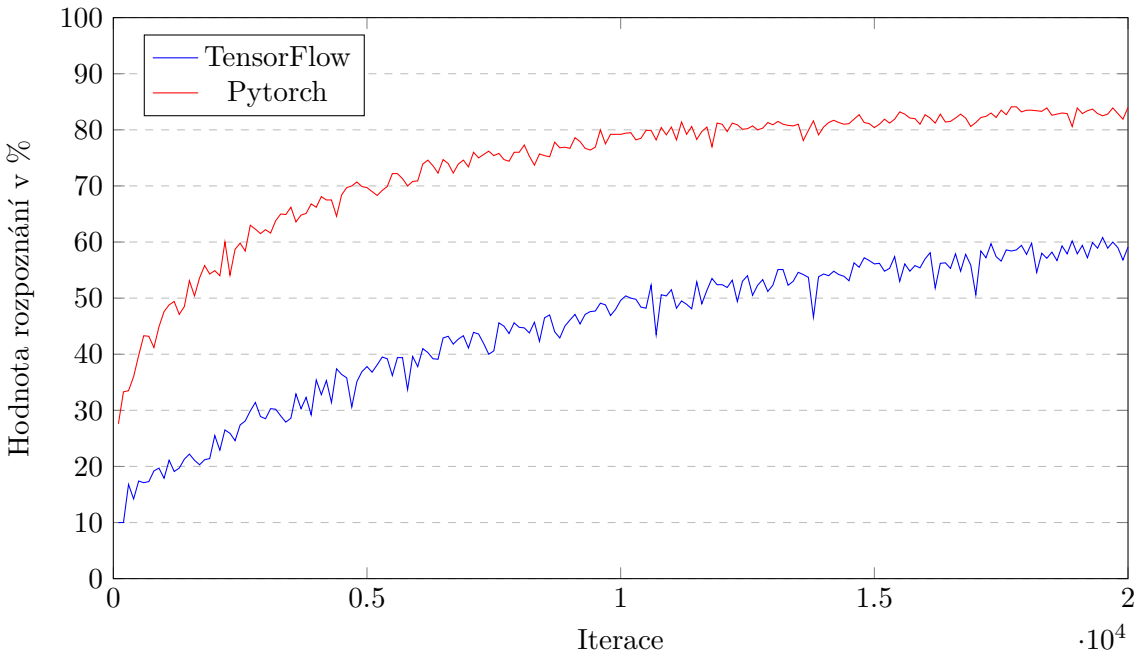
* nejlépe dosažený výsledek (celá tabulka viz příloha A.2)
ČI v s - čas iterace v sekundách,
HL - hodnota loss,
KVU - konečný výsledek učení.

Protože sítě mají jiné parametry, je potřeba srovnat vyhodnocení sítě i se stejnými parametry. Bylo nutné patřičně sítě upravit, a to v síti Pytorch byl upraven parametr $learning\_rate = 0.0001$, který je teď stejný jako v sítí TF a síť TensorFlow byla upravena trénovací parametr $Train\_batch\_size$ z hodnoty 64 na hodnotu 8. V rámci trénovaní byla síť Pytorch opět pomalejší a to s časem 9 hod, 1 min a 54 sec, ale s výsledkem 84.0% oproti TF, který byl schopný síť natrénovat pouhých 39 min a 26 sec a však s neuspokojivým výsledkem 59.1%. Časové zdržení u Pytorch s velké míry ovlivňuje i fakt, že po každých 100 iteracích vypisuje informační výpis, který musí otestovat aktuálně natrénovanou síť. Zatímco TF je schopné trénovat 256 najednou, Pytorch pouhých 8, což značně zpomaluje učení. Pokud by se výpis nevypisoval, dostali bychom mnohem lepší čas a to 2 hod, 59 min a 35 sec u sítě Pytorch a u TensorFlow sítě 21 min a 58 sec.

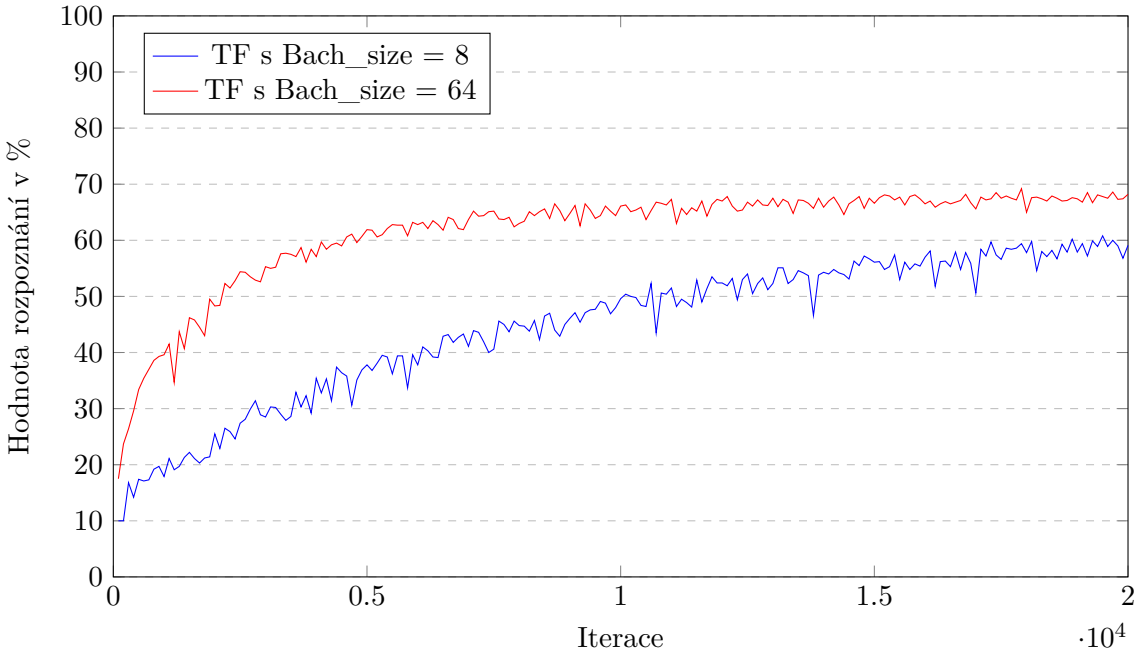
Ve srovnání v grafu 23 je zřetelné, jak změna hodnoty *batch_size* ovlivnila u sítě v TF hodnotu loss, která se kvůli méně vtékajícím datům mnohem hůře učila, oproti sítě Pytorch, které změna v učební konstantě ovlivnila velice málo (graf 26). Také srovnání úspěšnosti vyhodnocení se změnilo zatím co u PyToch došlo k menšímu zlepšení (graf 24), při změně učební konstanty u sítě TF změna počtu snímků v dávce úspěšnost sítě radikálně snížila (graf 25).



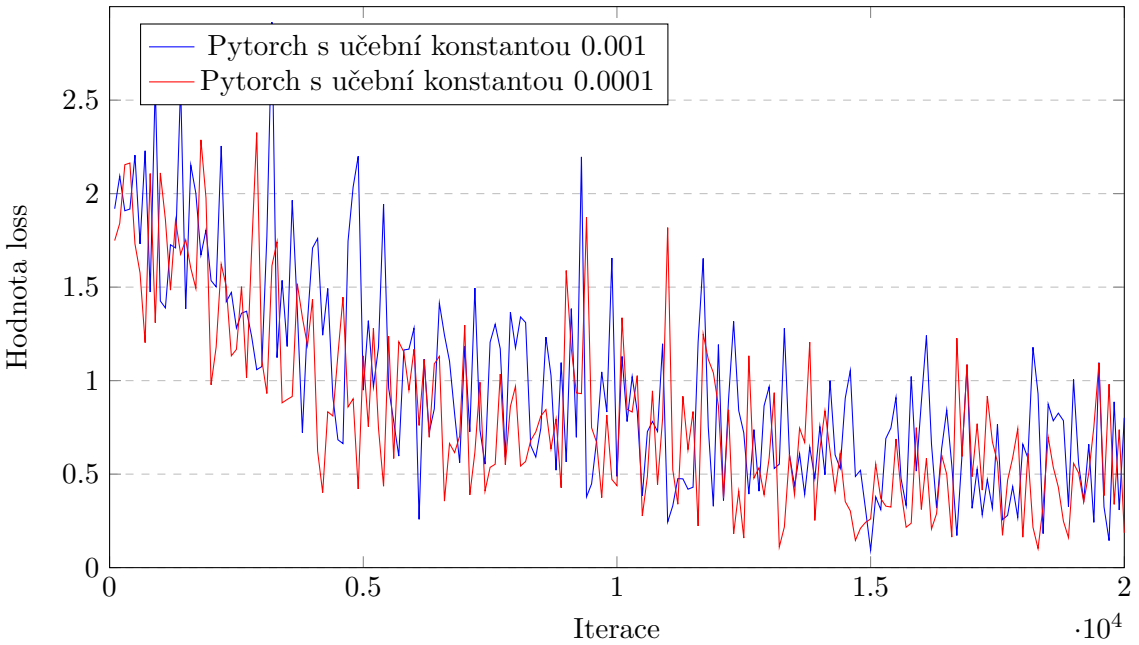
Graf 23: Srovnání hodnoty loss při učení se stejnými parametry.



Graf 24: Srovnání rozpoznání se stejnými parametry sítě.



Graf 25: Vliv při rozpoznání na síť TF při snížení snímků v dávce.



Graf 26: Vliv změny konstanty učení na síť v Pytorch.

10 Závěr

Cílem práce bylo seznámení s deep learning v analýze obrazu, což umožňují neuronové sítě. Jsou zde popsány funkce neuronové sítě a seznámení s nimi.

První kapitoly se zaměřují na samotný neuron v porovnání s neuronem v lidském mozku, a na strukturu umělého neuronu. Kde došlo ke zjištění podobnosti mezi lidským a umělým neuronem. Také jak se samotný umělý neuron tvoří a jak probíhá jeho učení.

V rámci další kapitoly se z neuronu tvoří neuronové sítě s popisem jednotlivých možných topologií neuronových sítí. V těchto kapitolách jsem zjistil, jak fungují vícevrstvé neuronové sítě, jeho modely a jejich tvorba.

Práce byla zaměřena na konvoluční neuronové sítě, jejich charakteristika, možnosti i samotná tvorba. V rámci této kapitoly jsem se naučil jak konvoluční síť funguje a jak ji efektivně vytvořit. Na závěr byly popisovány jednotlivé modely konvolučních sítí, jejich vznik a topologie.

Praktická část se zabývá tvorbou konvolučních neuronových sítí za pomocí volně dostupných frameworků. V úvodu jsou sepsány nejpoužívanější frameworky se stručnou charakteristikou použití. Dopodrobna se poté zabývá frameworkem TensorFlow od společnosti Google. Tvorba samotné sítě a implementace testovací sítě vyhodnocující dataset MNIST a dataset CIFAR10.

Další framework pro testovaní byl vybrán Pytorch, s jehož charakteristikou a tvorbou neuronové sítě se zabývá následující kapitola. V rámci ní je sepsáno vše od instalace, tvorby modelu až po trénovaní a následné vyhodnocování implementovaného modelu GoogLeNet.

Poslední kapitola pak tvoří samotné testování obou těchto sítí, kde jsou vyobrazeny rozdíly mezi oběma testovanými frameworky.

Díky této práci jsem získal povědomí o neuronových sítích a možnosti jejich implementací v rámci dvou různých frameworku. V testování jsem pak došel k výsledku, že každá síť je jedinečná a nikdy není možné docílit stejných rozpoznávacích hodnot při opakovaném trénování sítě. Ačkoli se zdá tvorba sítí v rozdílných frameworcích podobná, výrazně se liší v samotném přístupu sítě k datům a následně její vyhodnocení. I proto může dojít k lepším výsledkům u sítě, která k trénování využila méně dat, ale zpracovávala je stejnou nebo i delší dobu než síť, která využila větší množství dat.

Literatura

[1] *Chapter 10. Neural Networks* [online]. [cit. 2017-10-15]. Dostupné z: `http://natureofcode.com/book/chapter-10-neural-networks/`

[2] *Biologické algoritmy (4) - Neuronové sítě* [online]. 2012 [cit. 2017-10-15]. Dostupné z: https://www.root.cz/clanky/biologicke-algoritmy-4-neuronove-site/?ic=serial-boxicc=text-title

[3] *Matematický model a aktivní dynamika neuronu* [online]. [cit. 2017-10-15].Dostupné z: http://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat–umela-inteligence–neuronove-site-jednotlivy-neuron–jednotlivy-neuron–matematicky-model-a-aktivni-dynamika-neuronu

[4] KRAJČOVIČOVÁ, Mária. *Konvulční neurovnová síť pro zpracování obrazu.* Antonínská 548/1, 601 90 Brno-střed, 2015. Diplomová práce. Vysoké učení technice v Brně. Vedoucí práce Doc. Ing. RADIM BURGET, Ph.D.

[5] *Umělá inteligence I: Neuronové sítě* [online]. [cit. 2017-11-17]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/index.pl?cast=21471

[6] *Biologické algoritmy (5) - Neuronové sítě* [online]. [cit. 2018-01-03].Dostupné z: https://www.root.cz/clanky/biologicke-algoritmy-5-neuronove-site/?ic=serial-box&icc=text-title

[7] *Neuronové sítě.* In: Neuronové sítě [online]. [cit. 2018-01-03]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/download.pl?objekt=23134

[8] *Neuronové sítě* [online]. [cit. 2018-02-27]. Dostupné z: https://nlp.fi.muni.cz/uui/referaty2009/kabath$_d$*avid*/*referat.pdf*

[9] *Hopfield network* [online]. [cit. 2018-02-27]. Dostupné z: https://en.wikipedia.org/wiki/Hopfield$_n$*etwork*

[10] *Hopfieldova síť* [online]. [cit. 2018-02-27]. Dostupné z: http://neurony.wz.cz/program/CapacityHopf/CapacityHopfAppletCz2.html

[11] *Convolutional Neural Network* [online]. [cit. 2018-04-14]. Dostupné z: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[12] *Optické rozpoznávání ručně psaného textu* [online]. Český Brod, 2017 [cit. 2018-03-01]. STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST. Gymnázium Český Brod, Vítězná 616, 282 27 Český Brod.

[13] *Caffe: Deep learning* [online]. [cit. 2018-03-14]. Dostupné z: http://caffe.berkeleyvision.org/

[14] *TensorFlow* [online]. [cit. 2018-03-14]. Dostupné z: https://www.tensorflow.org/

[15] *Torch* [online]. [cit. 2018-03-14]. Dostupné z: http://torch.ch/

[16] *Caffe 2* [online]. [cit. 2018-03-14]. Dostupné z: https://developer.nvidia.com/caffe2

[17] *The Microsoft Cognitive Toolkit* [online]. [cit. 2018-04-14]. Dostupné z: https://docs.microsoft.com/en-us/cognitive-toolkit/

[18] *Apache MXNet* [online]. [cit. 2018-03-14]. Dostupné z: https://mxnet.apache.org/

[19] *Chainer* [online]. [cit. 2018-03-14]. Dostupné z: https://docs.chainer.org/en/stable/tutorial/basic.html

[20] *TensorFlow : Layers* [online]. [cit. 2018-03-14]. Dostupné z: https://www.tensorflow.org/tutorials/layers

[21] *Convolutional Neural Networks* [online]. [cit. 2018-04-14]. Dostupné z: http://cs231n.github.io/convolutional-networks/

[22] *Pytorch* [online]. [cit. 2018-03-14]. Dostupné z: http://Pytorch.org/about/

[23] *VÍCEVRSTVÁ NEURONOVÁ SÍŤ* [online]. Brno, 2013 [cit. 2018-03-22]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=64938. Bakalářská práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

[24] *Úvod do zpracování časově závislých dat neuronovými sítěmi* [online]. [cit. 2018-04-14]. Dostupné z: http://www.elektrorevue.cz/clanky/03031/index.html

[25] *Deep Learning v analýze obrazu* [online]. Ostrava, 2016 [cit. 2018-04-14]. Dostupné z: https://dspace.vsb.cz/handle/10084/115986. Diplomová práce. VŠB – Technická univerzita Ostrava.

[26] *Second Generation Neural Networks* [online]. [cit. 2018-04-14]. Dostupné z: https://www.mql5.com/en/articles/1103

[27] *Multi-Layer Neural Networks with Sigmoid Function* [online]. [cit. 2018-04-14]. Dostupné z: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

[28] *Konvoluční neuronové sítě* [online]. [cit. 2018-04-14]. Dostupné z: https://www.superlectures.com/openalt2015/konvolucni-neuronove-site

[29] *Tutorial 2: Creating and training a simple digit classifier* [online]. [cit. 2018-03-14]. Dostupné z: http://eblearn.sourceforge.net/beginner_tutorial2_train.html

[30] *CNNs Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more* [online]. [cit. 2018-03-14]. Dostupné z: https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[31] *The 9 Deep Learning Papers You Need To Know About* [online]. [cit. 2018-03-14]. Dostupné z: https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

[32] *Torch* [online]. [cit. 2018-03-14]. Dostupné z: https://i.stack.imgur.com/

[33] *A simple explanation of reverse-mode automatic differentiation* [online]. [cit. 2018-04-19]. Dostupné z: https://justindomke.wordpress.com/2009/03/24/a-simple-explanation-of-reverse-mode-automatic-differentiation/

[34] *The CIFAR-10 dataset* [online]. [cit. 2018-04-19]. Dostupné z: http://www.cs.toronto.edu/ kriz/cifar.html

[35] *THE MNIST DATABASE* [online]. [cit. 2018-04-19]. Dostupné z: http://yann.lecun.com/exdb/mnist/

[36] *Theano* [online]. [cit. 2018-04-19]. Dostupné z: http://deeplearning.net/software/theano/

A Přiložené soubory na CD

A.1 Bakalářská práce

Bakalářská práce ve formátu PDF. Ve složce `10Dokumenty/DeepLearning.pdf`

A.2 Tabulky testovacích dat

Tabulky dat z testovaní použité na grafy. Ve složce `10Dokumenty/TestovaciData.ods`

A.3 Testovací aplikace TensorFlow MNIST

Testovací aplikace s použitím frameworku TensorFlow a datasetu MNIST. Ve složce `20Aplication/tensorflow/TensorFlowGoogleNetMNIST.py`

A.4 Testovací aplikace TensorFlow CIFAR10

Testovací aplikace s použitím frameworku TensorFlow a datasetu CIFAR10. Ve složce `20Aplication/tensorflow/TensorFlowGoogleNetCIFAR.py`

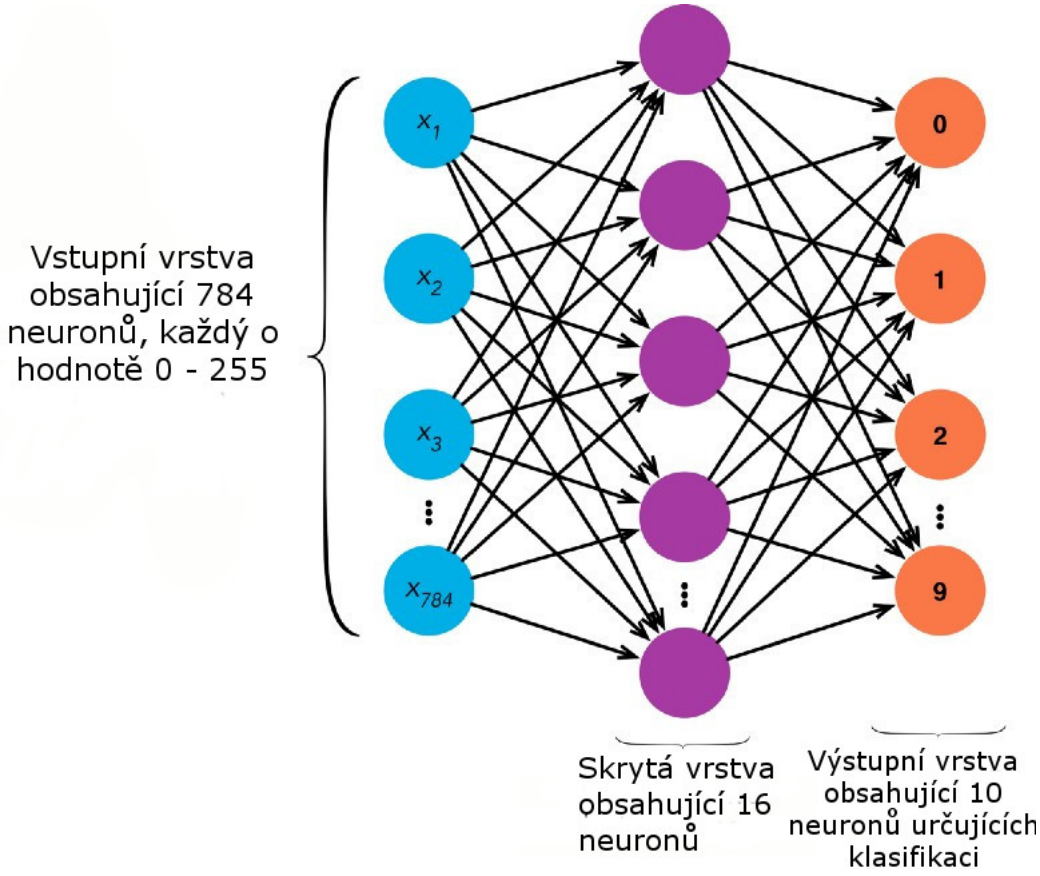
A.5 Testovací aplikace Pytorch MNIST

Testovací aplikace s použitím frameworku Pytorch a datasetu MNIST. Ve složce `20Aplication/pytorch/TorchGoogLeNetMNIST.py`

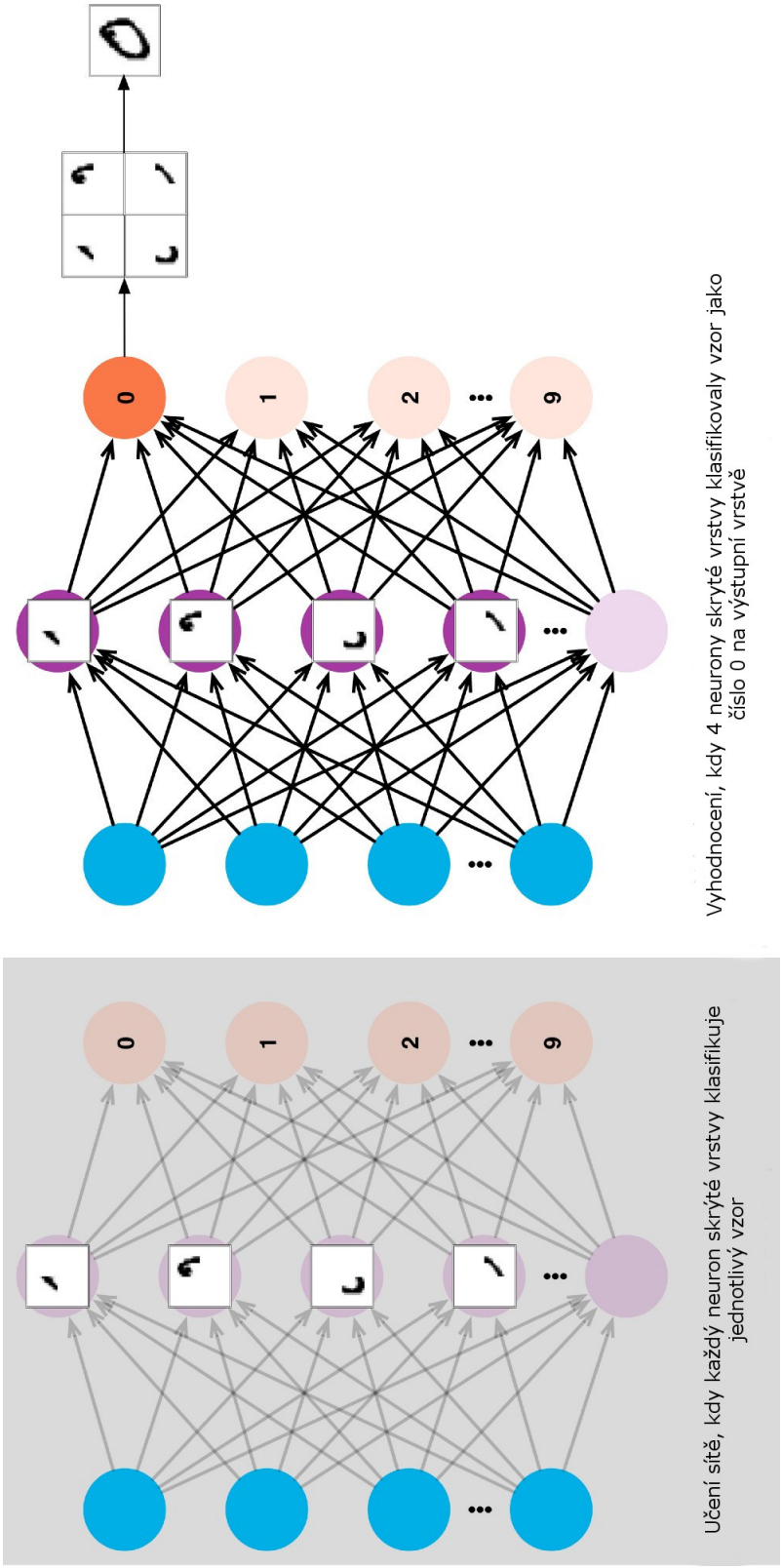
A.6 Testovací aplikace Pytorch CIFAR10

Testovací aplikace s použitím frameworku Pytorch a datasetu CIFAR10. Ve složce `20Aplication/pytorch/TorchGoogLeNetCIFAR.py`

B Model zpracování vícevrstvé neuronové sítě

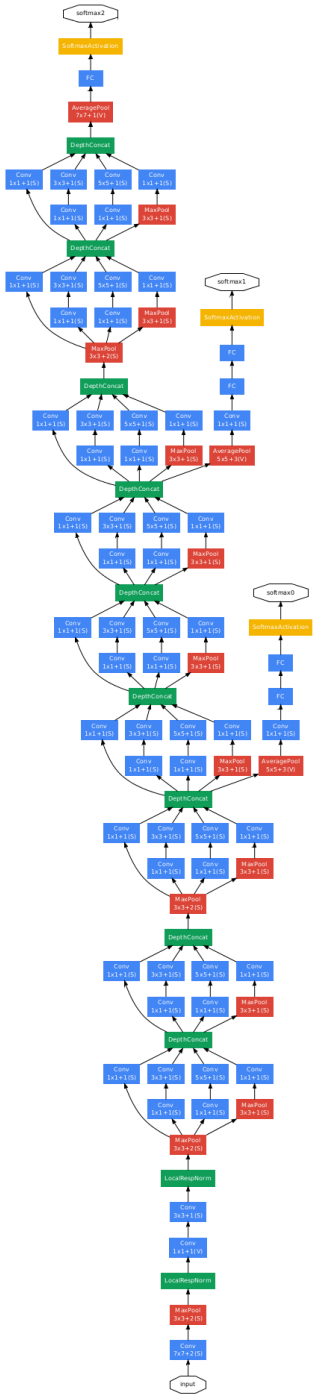


Obrázek 27: Vícevrstvá neuronová síť pro zpracovaní datasetu MNIST. [23]



Obrázek 28: Ukázka zpracování a vyhodnocení vícevrstvé neuronové sítě. [27]

C Model konvoluční sítě GoogleNet



Obrázek 29: Síť CNN GoogleNet Modrá - konvoluční vrstvy, červená - pooling vrstvy, žlutá - softmax, zelená - ostatní. [32]

D Tabulka srovnání frameworků

Tabulka 4: Srovnání frameworků.

	PČ v s	PHL	PV	CČU	KVU
1. testování					
TensorFlow	0.140133010524	1.883735204	25.92%	00:01:55	32.20%
Pytorch	0.544530010223	1.82498176098	24.73%	00:14:17	28.73%
2. testování					
TensorFlow	0.140062301011	1.843850303	25,20%	00:01:55	31.50%
Pytorch	0.544195413589	1.97078340054	25.40%	00:13:53	26.89%
3. testování					
TensorFlow	0.141323110877	1.851040292	26.60%	00:01:53	29.50%
Pytorch	0.537192869186	2.11794669628	19.40%	00:13:35	19.70%
4. testování					
TensorFlow	0.140332174301	1.89306271076	24,93%	00:01:56	32,50%
Pytorch	0.538019990921	2.11341662407	27.23%	00:13:34	30.43%
5. testování					
TensorFlow	0.139996910095	1.95901196003	22.90%	00:01:56	26.60%
Pytorch	0.533269262314	1.90548255444	26.80%	00:13:33	32.83%
6. testování					
TensorFlow	0.145836019516	1.91018667221	25.50%	00:02:01	28.60%
Pytorch	0.534822797775	2.21290411949	23.80%	00:13:35	24.50%
7. testování					
TensorFlow	0.139896202087	1.82496292591	24.30%	00:02:04	31.00%
Pytorch	0.535317993164	1.7505543232	24.40%	00:13:34	26.61%
8. testování					
TensorFlow	0.14845457077	2.02653262615	23.60%	00:01:58	28.30%
Pytorch	0.53708024025	2.06349446774	21.90%	00:13:36	23.26%
9. testování					
TensorFlow	0.139980173111	1.81850261688	26.10%	00:01:55	30.70%
Pytorch	0.536700391769	1.84071042538	25.10%	00:13:35	28.91%
10. testování					
TensorFlow	0.164199018478	1.89348950386	24.60%	00:01:58	33.20%
Pytorch	0.532991981506	1.89119608402	23.40%	00:13:33	30.40%
Pruměrné výsledky ze všech testování					
TensorFlow	0.1440155068	1.890437481	24.97%	00:01:57	30.41%
Pytorch	0.5374120951	1.969147046	24.33%	00:13:41	27.23%

PČ v s - průměrný čas v sekundách,
PHL - průměrná hodnota loss,
PV - průměrný výsledek,
CČU - Celkový čas učení,
KVU - Konečný výsledek učení.